

CMS Drupal 7

РУКОВОДСТВО ПО РАЗРАБОТКЕ СИСТЕМЫ УПРАВЛЕНИЯ ВЕБ-САЙТОМ

ТРЕТЬЕ ИЗДАНИЕ

PRO DRUPAL 7 DEVELOPMENT

THIRD EDITION

Todd Tomlinson

Apress®

CMS Drupal 7

РУКОВОДСТВО ПО РАЗРАБОТКЕ СИСТЕМЫ УПРАВЛЕНИЯ ВЕБ-САЙТОМ

ТРЕТЬЕ ИЗДАНИЕ

Тодд Томлинсон



Москва • Санкт-Петербург • Киев
2011

ББК 32.973.26-018.2.75
Т56
УДК 681.3.07

Издательский дом “Вильямс”
Зав. редакцией С.Н. Тригуб
Перевод с английского Ю.И. Корниенко, А.А. Моргунова
Под редакцией Ю.Н. Артеменко

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

Томлинсон, Тодд.

Т56 CMS Drupal 7: руководство по разработке системы управления веб-сайтом, 3-е изд. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2011. — 560 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1743-0 (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 APress, Berkeley, CA.

Authorized translation from the English language edition published by APress, Copyright © 2010 by Todd Tomlinson

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Russian language edition is published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2011.

Научно-популярное издание

Тодд Томлинсон

CMS Drupal 7: руководство по разработке системы управления веб-сайтом 3-е издание

Верстка Т.Н. Артеменко
Художественный редактор В.Г. Павлютин

Подписано в печать 07.07.2011. Формат 70×100/16.

Гарнитура Times. Печать офсетная.

Усл. печ. л. 49,02. Уч.-изд. л. 34,3.

Тираж 1500 экз. Заказ № 0000.

Отпечатано с готовых диапозитивов в ГУП “Типография «Наука»
19034, Санкт-Петербург, 9-я линия В. О., 12.

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

ISBN 978-5-8459-1743-0 (рус.)

ISBN 978-1-43-022838-7 (англ.)

© Издательский дом “Вильямс”, 2011

© by Todd Tomlinson, 2011

Оглавление

Предисловие	17
Введение	21
Глава 1. Как работает Drupal	23
Глава 2. Создание модулей	32
Глава 3. Хуки, действия и триггеры	48
Глава 4. Система меню	68
Глава 5. Работа с базами данных	94
Глава 6. Работа с пользователями	116
Глава 7. Работа с нодами	134
Глава 8. Работа с полями	155
Глава 9. Система тем	173
Глава 10. Работа с блоками	204
Глава 11. Интерфейс Form API	217
Глава 12. Манипулирование данными, вводимыми пользователем: система фильтров	265
Глава 13. Поиск и индексация контента	275
Глава 14. Работа с файлами	289
Глава 15. Работа с таксономией	306
Глава 16. Кэширование	323
Глава 17. Сеансы	335
Глава 18. Использование jQuery	344
Глава 19. Локализация и перевод	367
Глава 20. Протокол XML-RPC	393
Глава 21. Написание защищенного кода	404
Глава 22. Передовые приемы разработки	424
Глава 23. Оптимизация Drupal	436
Глава 24. Инсталляционные профили	462
Глава 25. Тестирование	480
Приложение А. Справочник по таблицам базы данных	497
Приложение Б. Ресурсы	540
Предметный указатель	547

Содержание

Предисловие	17
Об авторе	18
О технических рецензентах	19
Благодарности	20
Введение	21
От издательства	22
Глава 1. Как работает Drupal	23
Что собой представляет Drupal	23
Стек технологий	23
Ядро	24
Интерфейс администрирования	25
Модули	25
Хуки	25
Темы	27
Ноды	27
Поля	27
Блоки	28
Расположение файлов	28
Обслуживание запросов	30
Роль веб-сервера	30
Процесс начальной загрузки	30
Обработка запроса	30
Оформление данных	31
Резюме	31
Глава 2. Создание модулей	32
Создание файлов	32
Реализация хука	34
Добавление параметров для конкретного модуля	35
Определение собственного раздела администрирования	42
Вывод формы параметров пользователю	44
Проверка параметров, введенных пользователем	45
Хранение параметров	46
Использование таблицы <code>variables</code>	46
Извлечение сохраненных значений с помощью функции <code>variable_get()</code>	47
Дальнейшие шаги	47
Резюме	47
Глава 3. Хуки, действия и триггеры	48
События и триггеры	48
Действия	50
Пользовательский интерфейс триггера	50
Первое действие	52
Назначение действия	53
Изменение триггеров, поддерживаемых действием	54

Использование контекста в действиях	58
Подготовка контекста модулем триггера	59
Изменение существующих действий с помощью <code>drupal_alter()</code>	59
Формирование контекста	60
Хранение действий	62
Таблица <code>actions</code>	62
Идентификаторы действий	62
Непосредственный вызов действия с помощью <code>actions_do()</code>	62
Определение собственных триггеров с помощью <code>hook_trigger_info()</code>	63
Добавление триггеров в существующие хуки	66
Резюме	67
Глава 4. Система меню	68
Отображение обратных вызовов	68
Отображение URL-адресов на функции	68
Создание элемента меню	69
Аргументы обратного вызова страницы	73
Обратные вызовы страниц в других файлах	75
Добавление ссылки в блок навигации	76
Вложенные меню	77
Управление доступом	78
Локализация и настройка заголовка	79
Определение обратного вызова заголовка	80
Обобщенные символы в элементах меню	81
Основные обобщенные символы	81
Обобщенные символы и параметры обратного вызова страницы	82
Использование значения обобщенного символа	82
Обобщенные символы и замена параметров	83
Передача дополнительных аргументов функции загрузки	84
Специальные предопределенные аргументы загрузки <code>%map</code> и <code>%index</code>	85
Сборка путей из обобщенных символов с помощью функций <code>to_arg()</code>	85
Специальные варианты обобщенных символов и функции <code>to_arg()</code>	85
Изменение элементов меню из других модулей	86
Изменение ссылок меню из других модулей	87
Виды элементов меню	88
Часто встречающиеся задачи	90
Назначение обратных вызовов без добавления ссылки в меню	90
Вывод элементов меню в виде вкладок	90
Соккрытие существующих элементов меню	92
Использование <code>menu.module</code>	92
Распространенные ошибки	93
Резюме	93
Глава 5. Работа с базами данных	94
Определение параметров базы данных	94
Уровень абстракции баз данных	95
Подключение к базе данных	95
Выполнение простых запросов	97
Выборка результатов запроса	98
Получение одного значения	98
Получение нескольких строк	98

8 Содержание

Использование конструктора запросов и объектов запросов	99
Получение ограниченного диапазона результатов	100
Получение результатов для страничного вывода	100
Другие распространенные запросы	101
Вставки и обновления данных с помощью <code>drupal_write_record()</code>	102
API-интерфейс схемы	103
Использование файлов <code>.install</code>	103
Создание таблиц	103
Использование модуля схемы	106
Отображение типов полей из схемы в базу данных	106
Объявление конкретного типа столбца с помощью <code>mysql_type</code>	109
Обслуживание таблиц	110
Удаление таблиц при деинсталляции	111
Изменения существующих схем с помощью <code>hook_schema_alter()</code>	111
Изменение запросов других модулей с помощью <code>hook_query_alter()</code>	112
Подключение к нескольким базам данных в Drupal	113
Использование временной таблицы	114
Написание собственного драйвера СУБД	115
Резюме	115
Глава 6. Работа с пользователями	116
Объект <code>\$user</code>	116
Проверка, выполнил ли пользователь вход	118
Хуки пользователя	119
Функция <code>hook_user_view(\$account, \$view_mode)</code>	120
Процесс регистрации пользователя	122
Сбор информации о пользователях с помощью <code>profile.module</code>	124
Процесс входной регистрации	125
Добавление данных в объект <code>\$user</code> во время загрузки	125
Предоставление категорий пользовательской информации	128
Внешний вход	129
Резюме	133
Глава 7. Работа с нодами	134
Так что же собой представляет нода?	134
Не все является нодами	136
Создание модуля ноды	137
Хранение нод	149
Создание типа ноды с помощью собственных типов контента	150
Ограничение доступа к нодам	150
Что такое область	152
Что такое идентификатор разрешения	152
Резюме	154
Глава 8. Работа с полями	155
Создание типов контента	155
Добавление полей в тип контента	157
Создание произвольного поля	160
Программное добавление полей	170
Резюме	172

Глава 9. Система тем	173
Темы	173
Установка готовой темы	173
Создание темы	174
Файл .info	180
Добавление областей в тему	180
Добавление CSS-файлов в тему	181
Добавление файлов JavaScript	181
Добавление параметров в тему	181
Шаблонные файлы	184
Крупный план	184
Файл html.tpl.php	186
Файл page.tpl.php	187
Файл region.tpl.php	191
Файл node.tpl.php	191
Файл field.tpl.php	194
Файл block.tpl.php	195
Переопределение шаблонных файлов	197
Другие шаблонные файлы	197
Знакомство с функцией theme ()	198
Принцип действия функции theme ()	198
Переопределение элементов, допускающих темизацию	199
Переопределение с помощью шаблонных файлов	201
Добавление и манипулирование шаблонными переменными	202
Использование модуля разработчика тем	203
Резюме	203
Глава 10. Работа с блоками	204
Что такое блок?	204
Параметры настройки блоков	206
Размещение блока	207
Определение блока	207
Использование блочных хуков	209
Создание блоков	209
Активация блока при инсталляции модуля	215
Примеры видимости блоков	216
Отображение блока только для вошедших пользователей	216
Вывод блока только для анонимных пользователей	216
Резюме	216
Глава 11. Интерфейс Form API	217
Обработка формы	217
Инициализация процесса	219
Установка маркера	219
Установка идентификатора	219
Сбор всех возможных определений элемента формы	219
Поиск функции проверки достоверности	220
Поиск функции отправки	220
Разрешение модулям изменять форму перед ее построением	221
Построение формы	221

10 Содержание

Разрешение функциям изменять форму после ее построения	221
Проверка факта отправки формы	221
Обнаружение функции темы для формы	222
Разрешение модулям изменять форму перед ее визуализацией	222
Визуализация формы	222
Проверка достоверности формы	223
Отправка формы	224
Перенаправление пользователя	224
Создание базовых форм	224
Свойства формы	226
Идентификаторы формы	227
Наборы полей	227
Темизация форм	230
Спецификация функций проверки достоверности и отправки формы с помощью <code>hook_forms()</code>	233
Порядок вызова функций тем, проверки достоверности и отправки	234
Реализация функции проверки достоверности	234
Перестройка формы	237
Реализация функции отправки	238
Изменение форм с помощью <code>hook_form_alter()</code>	239
Программная отправка формы с помощью <code>drupal_form_submit()</code>	240
Динамические формы	240
Свойства Form API	247
Свойства для корневого элемента формы	247
Свойства, добавляемые ко всем элементам	248
Свойства, доступные во всех элементах	248
Элементы формы	251
Свойство <code>#ahah</code>	259
Резюме	264
Глава 12. Манипулирование данными, вводимыми пользователем: система фильтров	265
Фильтры	265
Фильтры и текстовые форматы	266
Инсталляция фильтра	268
Когда нужно использовать фильтры	268
Создание специального фильтра	270
Реализация <code>hook_filter_info()</code>	271
Функция обработки	271
Вспомогательная функция	272
Резюме	274
Глава 13. Поиск и индексация контента	275
Создание специальной страницы поиска	275
Форма поиска по умолчанию	276
Расширенная форма поиска	276
Дополнение к форме поиска	277
Использование HTML-индексатора поиска	280
Когда используется индексатор	280
Как работает индексатор	280
Резюме	288

Глава 14. Работа с файлами	289
Как Drupal обслуживает файлы	289
Управляемый и неуправляемый API-интерфейс Drupal	289
Общедоступные файлы	290
Приватные файлы	291
Параметры настройки PHP	291
Обработка медиа-данных	292
Поле загрузки	292
Видео и аудио	293
API-интерфейс для работы с файлами	293
Схема базы данных	294
Общие задачи и функции	295
Выяснение заданного по умолчанию URI файлов	295
Копирование и перемещение файлов	296
Проверка каталогов	296
Выгрузка файлов	296
Получение URL для файла	300
Поиск файлов в каталоге	301
Поиск временного каталога	301
Нейтрализация опасных файлов	302
Проверка дискового пространства	304
Хуки аутентификации для загрузки	304
Резюме	305
Глава 15. Работа с таксономией	306
Структура таксономии	306
Создание словаря	307
Создание терминов	307
Назначение словаря типу контента	307
Виды таксономии	308
Одноуровневый словарь	308
Иерархический словарь	308
Множественная иерархия	309
Просмотр контента по термину	310
Использование операций AND и OR в URL	311
Определение глубины для иерархических словарей	311
Автоматические RSS-каналы	312
Хранение таксономий	312
Словари на основе модулей	314
Создание словаря на основе модуля	314
Уведомление об изменениях в словаре с помощью хуков таксономии	314
Общие задачи	316
Отображение терминов таксономии, связанных с нодой	316
Построение собственных запросов таксономии	316
Использование функции <code>taxonomy_select_nodes()</code>	317
Функции таксономии	317
Извлечение информации о словарях	317
Добавление, изменение и удаление словарей	317
Извлечение информации о терминах	318
Добавление, изменение и удаление терминов	319

12 Содержание

Извлечение информации об иерархии терминов	320
Поиск нод с определенными терминами	322
Дополнительные ресурсы	322
Резюме	322
Глава 16. Кэширование	323
Когда необходимо кэширование	323
Как работает кэширование	324
Как кэширование используется в ядре Drupal	325
Система меню	326
Кэширование фильтрованного текста	326
Переменные администрирования и настройки модуля	326
Отключение кэширования	327
Кэширование страниц	328
Кэширование статической страницы	328
Кэширование данных с помощью <code>cache_set()</code>	331
Извлечение кэшируемых данных с помощью <code>cache_get()</code> и <code>cache_get_multiple()</code>	332
Использование функции <code>cache_is_empty</code> для проверки, пуст ли кэш	332
Очистка кэша с помощью <code>cache_clear_all</code>	332
Резюме	334
Глава 17. Сеансы	335
Что собой представляют сеансы	335
Использование сеансов	335
Настройки, связанные с сеансом	337
Файл <code>.htaccess</code>	337
Файл <code>settings.php</code>	337
Файл <code>bootstrap.inc</code>	338
Требование поддержки cookie-наборов	338
Хранилище	339
Жизненный цикл сеанса	340
Сеансовые диалоги	341
Первое посещение	341
Второе посещение	341
Пользователь с учетной записью	341
Общие задачи	342
Изменение времени истечения срока действия cookie-набора	342
Изменение имени сеанса	342
Хранение данных в сеансе	342
Резюме	343
Глава 18. Использование jQuery	344
Что собой представляет jQuery	344
Устаревший метод	345
Как работает jQuery	346
Использование селектора идентификации CSS	346
Использование селектора класса CSS	346
jQuery в Drupal	347
Ваш первый код jQuery	347
Выбор элемента по идентификатору	349

Объединение методов в цепочку	350
Добавление или удаление класса	350
Помещение существующих элементов в оболочку	350
Изменение значений элементов CSS	351
Куда помещать код JavaScript	351
Переопределяемый код JavaScript	354
Создание виджета для голосования с помощью jQuery	357
Построение модуля	359
Использование <code>Drupal.behaviors</code>	364
Способы расширения модуля	365
Совместимость	365
Дальнейшие шаги	366
Резюме	366
Глава 19. Локализация и перевод	367
Активация модуля <code>locale</code>	367
Перевод пользовательского интерфейса	367
Строки	367
Переопределения строк в <code>settings.php</code>	369
Замена строк с помощью модуля локализации	371
Экспорт перевода	376
Начало нового перевода	377
Использование командной строки	378
Использование модуля извлечения с веб-интерфейсом	378
Установка перевода на определенный язык	379
Поддержка языков с письмом справа налево	381
Согласование языков	382
Опция Default (По умолчанию)	383
Опция User (Пользовательский)	383
Глобальный объект <code>\$language</code>	384
Опция Path prefix only (Только префикс пути)	384
Опция Path prefix with language fallback (Префикс пути с обходом языка)	386
Опция URL Only (Только URL)	386
Перевод контента	387
Начальное знакомство с модулем перевода контента	387
Многоязыковая поддержка	387
Многоязыковая поддержка с переводом	388
Файлы, относящиеся к локализации и переводу	391
Дополнительные ресурсы	392
Резюме	392
Глава 20. Протокол XML-RPC	393
Что собой представляет XML-RPC	393
Предварительные требования для протокола XML-RPC	393
Клиенты XML-RPC	394
Пример клиента XML-RPC: получение текущего времени	394
Пример клиента XML-RPC: получение названия штата	395
Обработка ошибок клиента XML-RPC	396
Простой сервер XML-RPC	398
Отображение метода с помощью <code>hook_xmlrpc()</code>	399
Автоматическая проверка типа параметра с помощью <code>hook_xmlrpc()</code>	400

14 Содержание

Встроенные методы XML-RPC	400
system.listMethods	400
system.methodSignature	402
system.methodHelp	403
system.getCapabilities	403
system.multiCall	403
Резюме	403
Глава 21. Написание защищенного кода	404
Обработка вводимой пользователем информации	404
Соображения по поводу типов данных	405
Использование функций <code>check_plain()</code> и <code>t()</code> для очистки вывода	407
Использование функции <code>filter_xss()</code> для предотвращения межсайтовых атак с помощью сценариев	409
Использование функции <code>filter_xss_admin()</code>	411
Безопасная обработка URL	411
Обеспечение безопасности запросов с помощью функции <code>db_query()</code>	412
Сохранение конфиденциальности личных данных с помощью функции <code>hook_alter_query()</code>	414
Динамические запросы	415
Разрешения и обратные вызовы страницы	415
Подделки межсайтовых запросов (CSRF)	416
Безопасность файлов	416
Права доступа к файлу	416
Защищенные файлы	417
Загрузка файлов на сервер	418
Имена файлов и пути	418
Кодирование заголовков почты	418
Файлы для рабочей среды	419
Поддержка SSL	420
Автономный PHP	420
Безопасность AJAX и атаки повторением запроса	421
Безопасность API-интерфейса форм	422
Защита учетной записи суперпользователя	422
Резюме	423
Глава 22. Передовые приемы разработки	424
Стандарты кодирования	424
Выравнивание строк и пробелы	424
Операции	424
Приведение типов	425
Управляющие структуры	425
Вызовы функций	425
Объявления функций	426
Имена функций	426
Вызовы конструктора класса	427
Массивы	427
Кавычки	427
Конкатенация строк	428
Комментарии	428
Примеры документирования	429

Документирование констант	429
Документирование функций	429
Документирование реализаций хуков	431
Включение кода	431
Дескрипторы PHP-кода	431
Символы точки с запятой	432
URL-адреса примеров	432
Соглашения по именованию	432
Программная проверка стиля кодирования	432
Использование сценария <code>code-style.pl</code>	432
Использование модуля кодировщика	433
Поиск в коде с помощью <code>grep</code>	434
Резюме	435
Глава 23. Оптимизация Drupal	436
Кэширование — ключ к высокой производительности Drupal	436
Оптимизация PHP	438
Установка файла кэша кодов операций PHP в <code>/dev/zero</code>	439
Настройки пула процессов PHP	439
Настройка Apache	441
Модуль <code>mod_expires</code>	441
Перенос директив из <code>.htaccess</code> в <code>httpd.conf</code>	441
Сравнение предварительного ветвления MPM с рабочей конфигурацией Apache MPM	442
Балансировка размера пула Apache	443
Снижение длительности тайм-аута Apache	443
Отключение неиспользуемых модулей Apache	443
Использование Nginx вместо Apache	443
Использование Pressflow	444
Использование Varnish	444
Нормализация входящих запросов для увеличения коэффициента попаданий в кэш Varnish	445
Varnish: обнаружение внешних cookie-наборов	446
Модуль Boost	446
Сравнение Boost и Varnish	447
Настройка системы Linux для серверов с высоким уровнем трафика	447
Использование быстрых файловых систем	448
Сравнение выделенных и виртуальных серверов	449
Предотвращение обращения к внешним веб-службам	449
Уменьшение интервалов ожидания сервера	450
Оптимизация базы данных	450
Включение кэша запросов в MySQL	450
Производительность InnoDB MySQL в среде Windows	451
Производительность Drupal	451
Предотвращение ошибок 404	451
Отключение неиспользуемых модулей	452
Специфичные для Drupal методы оптимизации	452
Кэширование страниц	452
Оптимизация полосы пропускания	452
Усечение таблицы сеансов	452
Управление трафиком аутентифицированных пользователей	453

16 Содержание

Протоколирование в базе данных	453
Протоколирование в журнале операционной системы	453
Запуск cron	453
Архитектуры	454
Единственный сервер	454
Отдельный сервер базы данных	454
Отдельный сервер базы данных и кластер веб-серверов	454
Балансировка загрузки	455
Загрузки файлов на сервер и синхронизация	455
Использование разделяемой, монтируемой файловой системы	455
За рамками единственной файловой системы	456
Множество серверов базы данных	456
Репликация баз данных	456
Секционирование баз данных	456
Обнаружение узкого места	456
Перегрузка центрального процессора веб-сервера	457
Нехватка оперативной памяти на веб-сервере	458
Идентификация ресурсоемких запросов базы данных	459
Идентификация ресурсоемких страниц	459
Идентификация ресурсоемкого кода	459
Оптимизация таблиц	460
Кэширование запросов вручную	460
Изменение типа таблицы с MyISAM на InnoDB	460
Резюме	461
Глава 24. Инсталляционные профили	462
Создание нового инсталляционного профиля	462
Файл enhanced.info	463
Файл enhanced.profile	464
Файл enhanced.install	464
Использование хуков hook_install_tasks и hook_install_tasks_alter	478
Резюме	479
Глава 25. Тестирование	480
Настройка тестовой среды	480
Способ определения тестов	484
Тестовые функции	490
Тестовые утверждения	493
Резюме	496
Приложение А. Справочник по таблицам базы данных	497
Приложение Б. Ресурсы	540
Предметный указатель	547

Предисловие

Почти четыре года тому назад я написал предисловие к первому изданию этой книги. Все это время не хватало книги для разработчиков Drupal. С тех пор данная книга сделала огромный вклад в устойчивое развитие Drupal. Не думаю, что есть много разработчиков Drupal, которые бы не имели экземпляра этой книги.

Благодаря своей открытой природе, система Drupal стала намного более успешной, чем представлялось поначалу. Что не изменилось — так это здоровое стремление сообщества разработчиков Drupal к инновациям, быстрая реакция на постоянно меняющийся ландшафт веб-разработки и предоставление разработчикам веб-приложений почти бесконечной гибкости. Изменения в сообществе Drupal постоянны и являются ключом к нашему успеху.

Главный успех системе Drupal принесла версия Drupal 6. Тем не менее, буквально со дня появления Drupal 6 почти три года тому назад мы интенсивно работаем над версией Drupal 7. Свыше 800 участников написали исправления, которые были включены в ядро Drupal 7. Новая версия претерпела одно из наиболее крупных архитектурных изменений в истории развития Drupal, будет поставляться с множеством улучшений в API-интерфейсе и сможет поддерживать более крупные сайты, нежели ранее. Конечный результат в том, что версия Drupal 7 является даже лучшей платформой для разработки веб-приложений, чем Drupal 6, и это станет движущей силой развития Drupal в ближайшие годы.

Все эти изменения в системе также означают, что предыдущие издания книги устарело. К счастью, в третьем издании учтены последние усовершенствования Drupal. В книге раскрыты все возможности и средства Drupal 7 и описана внутренняя инфраструктура и проектные решения, положенные в основу версии Drupal 7.

Вооружившись этой книгой и копией исходного кода Drupal, вы получаете все, что необходимо для того, чтобы стать экспертом в области Drupal. Если в ходе работ вы обнаруживаете, как что-либо сделать лучше, с меньшим объемом кода или более элегантно и быстро, чем уже реализовано, вливайтесь в сообщество разработчиков и помогайте нам совершенствовать Drupal! Мне нравится пересматривать и принимать ваши исправления ядра Drupal, и я уверен — то же самое можно сказать о большинстве участников.

*Дрис Байтаерт,
основатель и руководитель проекта Drupal*

Об авторе

Тодд Томлинсон — вице-президент eGovernment Solutions в корпорации ServerLogic Corporation (Портленд, шт. Орегон). Свыше 15 лет основное внимание Тодд уделяет проектированию, разработке, развертыванию и поддержке сложных веб-решений для клиентов из всех секторов экономики по всему миру. Он использовал Drupal в качестве основной платформы для создания многофункциональных сайтов, таких как <http://arapahoelibraries.org/ald/>.

До корпорации ServerLogic Corporation Тодд занимал пост начальника отдела eBusiness Strategic Services в Oracle Corporation, помогая крупнейшим клиентам Oracle в стратегическом планировании по использованию Интернета как ключевого компонента их бизнес-деятельности. Он также бывший вице-президент Internet Solutions в Claremont Technology Group, вице-президент и технический директор в Emerald Solutions, управляющий директор в CNF Ventures и ведущий менеджер в Andersen Consulting/Accenture. Тодд имеет степень бакалавра в области вычислительной техники и диплом MBA, а также пишет диссертацию на соискание доктора философии.

Увлечение Drupal у Тодда проявляется в его навязчивой идее по евангелизации платформы и в его энтузиазме в разговорах с клиентами о возможности того, что они могут сделать с помощью Drupal. Чтобы оценить, насколько он “горит” идеей Drupal, попробуйте остановить его на улице и спросить: “Что такое Drupal и что эта система может сделать для меня?” Тодд является автором книги *Beginning Drupal 7*, выпущенной издательством Apress.

О технических рецензентах

Джошуа Брауэр попал во Всемирную Паутину еще студентом, специализирующимся на технической журналистике, когда увлекся вопросами систем управления контентом в 1995 г. Став членом сообщества Drupal в 2003 г., Джошуа занимался запуском сайтов Drupal самых разных размеров. В 2007 г. он ушел из IT-отдела, чтобы полностью сконцентрироваться на Drupal.

Джошуа — один из лидеров группы Boise Drupal Users Group и его часто можно найти выступающим на конференциях, учебных лагерях, местных встречах, а также в других местах, где людям интересно послушать о Drupal. Он много пишет на тему Drupal, и с его работами можно ознакомиться на сайте <http://joshuabrauer.com>.

В компании Acquia, Inc. Джошуа работает с заказчиками сайтов разного масштаба, от небольших до корпоративных, на всех этапах разработки, начиная с предварительного планирования и заканчивая хостингом и управлением операциями. Он черпает вдохновение из сообщества Drupal и его многочисленных достижений. Когда Джошуа отключен от Интернета, его можно найти с фотоаппаратом, наслаждающимся бесконечным разнообразием красивых мест на нашей планете.

Роберт Дуглас — ведущий консультант в компании Acquia, Inc., постоянный член ассоциации Drupal Association и один из основателей Die Drupal-Initiative, Drupal-ориентированной некоммерческой организации в Германии. Он занимается сопровождением модулей, участвует в разработке ядра и часто выступает с докладами на конференциях, посвященных Drupal. В издательстве Apress Роберт принимал участие в следующих проектах: *Building Online Communities with Drupal, phpBB, and WordPress* (автор, 2005 г.), *Pro Drupal Development* (технический рецензент, 2007 г.) и *Pro Drupal Development, Second Edition* (технический рецензент, 2008 г.).

Питер М. Воланин начал работать с Drupal в 2005 г., когда один из его друзей, сторонник Говарда Дина, вовлек его в проект по построению нового веб-сайта местного клуба Демократической партии, который разрабатывался на бета-версии Drupal 4.7. Очень скоро Питер заинтересовался исправлением ошибок и добавлением функциональных средств в ядро Drupal, а также созданием собственных модулей, идеи которых он черпал из повседневной работы над сайтом. Он стал известным участником в Drupal 5, 6 и 7 и членом команды написания документации по Drupal. Питер присоединился к команде, занимающейся обеспечением безопасности Drupal, а в 2010 г. был избран постоянным членом Drupal Association. В 2008 г. он присоединился к команде инженеров Acquia, Inc. и с тех пор наслаждается компанией своих звездных коллег. Ранее Питер с отличием окончил Принстонский университет, получил докторскую степень по физике в Университете штата Мичиган и занимался производственными исследованиями после защиты докторской диссертации в области биофизики и молекулярной биологии.

Благодарности

Бет — за бесконечную поддержку, поощрение, любовь и смех; спасибо за возвращение способности мечтать о большом будущем.

Моим дочерям, Анне, Алисе и Эмме — за то, что дали мне много часов времени, пока я писал эту книгу.

Моим родителям — за то, что предоставили мне средства, необходимые мне для путешествий.

Моей бабушке — за то, что вселила во мне интерес к авторству.

Дриссу — без вашего видения и страсти к платформе эта книга вряд ли увидела бы свет.

Команде из Acquia, Inc. — за поддержку во время написания сложных разделов книги, за преданность платформе и за неустанные усилия по запуску Drupal 7.

Тысячам разработчиков, которые принимали участие в развитии платформы до ее нынешнего состояния.

Моим клиентам — за оценки технологии и мнения по поводу того, что еще можно было сделать с ее помощью.

Джейсону, Дарену, Кэтрин и Стиву, моим коллегам по команде — за претворение в жизнь моих самых смелых ожиданий.

Введение

За свой относительно короткий период существования система Drupal оказала огромное влияние на “ландшафт” Интернета. Как система управления веб-контентом, Drupal сделала возможным создание веб-сайтов с развитой функциональностью и содержимым для крупных и мелких организаций. Как платформа для разработки веб-приложений, Drupal изменяет способ восприятия процесса построения веб-приложений. Когда я испытывал мощь платформы Drupal в первый раз, мне было известно, что она представляла собой нечто большее, чем просто еще одно решение для управления контентом. Когда я увидел, насколько легко и быстро можно было построить многофункциональный веб-сайт, я сфокусировал все внимание, равно как и карьеру на Drupal.

Мне часто задают вопрос: “Что такое Drupal?” Короткий ответ может быть сформулирован так: Drupal — это система управления веб-контентом с открытым исходным кодом, которая позволяет быстро и легко создавать простые и сложные веб-сайты, охватывающие весь спектр функций, от простого блога на корпоративном веб-сайте до веб-сайта социальной сети или практически чего-угодно. То, что вы можете построить с Drupal, ограничено только вашей фантазией, временем, которое понадобится уделить платформе, и знаниями возможностей Drupal — именно это и положено в основу настоящей книги.

По причине открытости кода сообщество Drupal постоянно совершенствует платформу и расширяет функциональность ядра, создавая новые и интересные дополнительные модули. Как только в Интернете появляется новая концепция, скорее всего, в течение нескольких дней будет создан новый модуль Drupal, который откроет к ней доступ. Именно работа сообщества делает платформу Drupal тем, чем она является сегодня, и тем, чем она станет завтра. В книге будет показано, как использовать функциональные средства, добавленные сообществом, что позволит легко и просто строить развитые решения с минимальными усилиями.

Сам факт выбора этой книги является первым шагом в вашем путешествии по пути обучения использованию Drupal. Простудировав материал, предложенный в книге, вы получите знания и обретете навыки для построения сложных и мощных веб-сайтов на основе Drupal. Вы также заложите фундамент для выхода за рамки основ и расширения концепций, раскрытых в этой книге.

Обучение Drupal подобно изучению любой новой технологии. Вы будете сталкиваться с определенными трудностями, которые могут заставить вернуться назад и заново перечитать материал. Надеюсь, что книга поможет сгладить все неровности и предоставит достаточно информации, чтобы легко преодолеть все препятствия. Я с нетерпением жду встречи с вашими работами в Интернете и надеюсь столкнуться с вами на очередной конференции DrupalCon.

Напоследок хочу дать несколько легкомысленный совет: учитесь заимствовать! После того как все фрагменты мозаики собраны воедино, мало что осталось придумать нового. Каждая новая идея, которую вы откроете, является просто комбинацией старых идей. Но сами-то идеи бесплатны! Любое полезное средство, которое обсуждалось на ТВ-шоу, было презентовано в брошюрах или на веб-сайтах коммерческих компаний, может быть принято, адаптировано и реализовано с приложением очень небольших усилий. Автоматизированный дом превращается в “умный” дом, а затем и в персонализированный “умный” дом.

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-сервер 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 координаты:

E-mail: info@williamspublishing.com

WWW: <http://www.williamspublishing.com>

Информация для писем из:

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

Украины: 03150, Киев, а/я 152

ГЛАВА 1

Как работает Drupal

В этой главе приведен краткий обзор технологии Drupal. Более подробные сведения относительно того, как работает каждая часть системы, можно найти в последующих главах. Здесь мы рассмотрим комплекс технологий, на которых работает Drupal, набор файлов, составляющих Drupal, и различные термины, используемые в технологии Drupal, такие как *ноды*, *хуки*, *блоки* и *темы*.

Что собой представляет Drupal

Drupal используется для создания веб-сайтов. Это модульная платформа с открытым исходным кодом, предназначенная для совместного управления содержимым (контентом). Она легко поддается расширению, соответствует большинству стандартов, ее код прост, понятен и невелик по объему. Система Drupal устанавливается в варианте, обеспечивающем выполнение лишь ее основных функций, а дополнительные возможности реализуются встроенными и сторонними модулями. Система Drupal допускает настройку, но только с помощью полной перезаписи ядра или добавления модулей, а не за счет изменения кода ядра. Кроме того, она эффективно отделяет управление контентом от его представления пользователям.

С помощью Drupal можно создать Интернет-портал, персональный, ведомственный или корпоративный веб-сайт, сайт Интернет-магазина, каталог сетевых ресурсов, Интернет-газету, сайт социальной сети, галерею изображений, интрасеть и практически любую другую разновидность веб-сайта.

Специальная группа, отвечающая за безопасность, стремится обеспечить защиту Drupal от новых угроз, выпуская обновления для системы безопасности. Некоммерческая организация под названием “Ассоциация Drupal” поддерживает Drupal, улучшая инфраструктуру веб-сайта drupal.org и организуя конференции и мероприятия, посвященные Drupal. А непрерывно пополняющееся сетевое сообщество пользователей, администраторов сайтов, проектировщиков и веб-разработчиков упорно трудится над улучшением программного обеспечения Drupal; см. <http://drupal.org> (<http://drupal.ru>) и <http://groups.drupal.org>.

Стек технологий

Система Drupal спроектирована как для работы на недорогих веб-серверах, обеспечивающих мало затратный хостинг, так и для больших распределенных сайтов. Первое означает, что должны использоваться наиболее популярные технологии, второе — что программный код должен быть тщательно отработан. Стек технологий Drupal представлен на рис. 1.1.

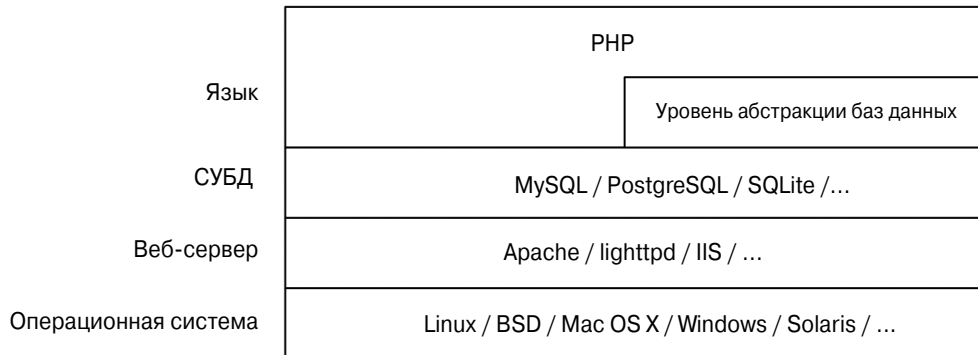


Рис. 1.1. Стек технологий Drupal

Операционная система в этом стеке находится на таком низком уровне, что Drupal почти не обращает на нее внимания. Drupal успешно работает в любой операционной системе, поддерживающей PHP.

Для работы с Drupal обычно применяется веб-сервер Apache, хотя могут использоваться и другие веб-серверы (в том числе Microsoft IIS). По этой причине Drupal поставляется с файлами `.htaccess`, которые обеспечивают защиту инсталляции Drupal. Чистые URL, т.е. URL без вопросительных знаков, амперсандов и других необычных символов, получаются с помощью Apache-компонента `mod_rewrite`. Это особенно важно, потому что при переходе к Drupal от другой системы управления содержимым или переносе статических файлов не приходится изменять их URL. Неизменяемые унифицированные идентификаторы ресурса (URI) — это “круто”, считает Тим Бернерс-Ли (<http://www.w3.org/Provider/Style/URI>). Чистые URL теперь доступны на других веб-серверах с помощью средств перезаписи URL этих веб-серверов.

Drupal взаимодействует со следующим уровнем стека (базой данных) через облегченный уровень абстракции баз данных, полностью переписанный в Drupal 7. Интерфейс для баз данных предоставляет API-интерфейс на основе модели объектов данных PHP (PHP data objects — PDO) и позволяет Drupal поддерживать любые СУБД, которые поддерживают PHP. Чаще всего используются СУБД MySQL и PostgreSQL, а в Drupal 7 — также и SQLite.

Drupal написана на языке PHP со строгим соблюдением стандартов (<http://drupal.org/nodes/318>) и, будучи системой с открытым кодом, постоянно пересматривается. Для Drupal простота изучения PHP означает легкость включения в процесс разработки новых участников, а пересмотр кода гарантирует, что эта легкость доступа не нанесет ущерба качеству конечного продукта. Отклики же, полученные новичками от сообщества, помогут им повысить свою квалификацию. Для Drupal 7 требуется PHP версии 5.2.

Ядро

Основу Drupal составляет небольшое по объему *ядро*. Именно его вы загружаете с сайта drupal.org. Ядро отвечает за выполнение основных функций, необходимых для обеспечения работоспособности других частей системы.

Ядро содержит код, который позволяет системе Drupal загружаться при поступлении запроса, библиотеку общих функций, часто используемых с Drupal, и модули, которые обеспечивают основные функциональные возможности, наподобие управления пользователями, таксономии и шаблонов (рис. 1.2).



Рис. 1.2. Основные компоненты ядра Drupal (не все)

Ядро также содержит основные функциональные блоки для большинства веб-сайтов, в том числе агрегирование потоков новостей, блоги, опросы и форумы.

Интерфейс администрирования

Интерфейс администрирования в Drupal тесно интегрирован с остальной частью сайта. Все функции администрирования легко доступны через административное меню, которое появляется в верхней части страницы, если войти от имени администратора сайта.

Модули

Drupal — действительно модульная среда. Ее функции реализуются с помощью *модулей*, использование которых можно разрешить или запретить. Новые возможности можно добавить на веб-сайт Drupal, включив имеющиеся и установив модули, написанные членами сообщества Drupal, или разработав новые модули. Таким образом, веб-сайты без особых запросов могут работать сразу, а в специализированные сайты можно добавить особые функции (рис. 1.3).

Модули расширяют Drupal, добавляя новые типы контента — рецепты, статьи блогов или файлы — и поведение, подобное почтовым уведомлениям, одноранговой публикации и агрегации. В Drupal используется шаблон проектирования “Инверсия управления” (Inversion of Control), где функции модулей вызываются средой в нужное время. Эти возможности модулей выполнять свои функции называются *хуками*.

Хуки

Хуки (hook — крючок, ловушка) можно считать внутренними событиями Drupal. Их также называют *обратными вызовами* (callbacks), хотя это не совсем верно. Поскольку они создаются в соответствии с соглашениями об именовании функций, а не при регистрации слушателя (listener), на самом деле обратными вызовами они не являются. Хуки позволяют модулям “вмешаться” в то, что происходит в остальной части Drupal.

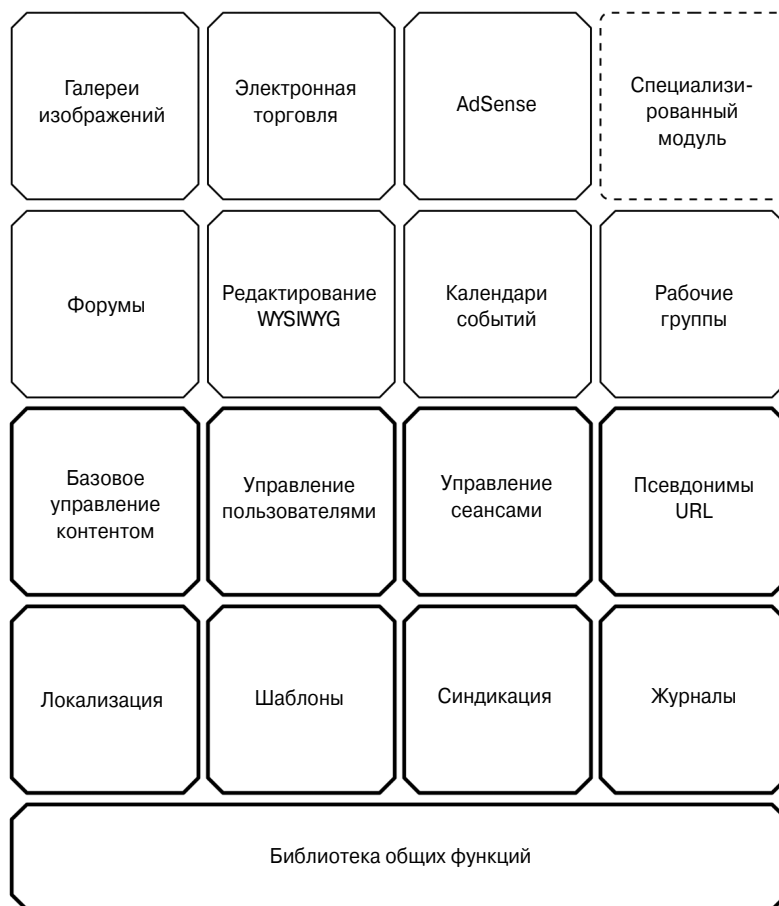


Рис. 1.3. Дополнительные модули, расширяющие возможности системы

Предположим, что пользователь регистрируется на веб-сайте Drupal. В момент входной регистрации пользователя Drupal запускает хук `hook_user_login`. Это означает, что будет вызываться любая функция с именем, образованным из имени модуля и имени хука. Например, это может быть функция `comment_user_login()` из модуля `comment`, `locale_user_login()` из модуля `locale`, `node_user_login()` из модуля `node` или любая другая функция с аналогичным именем. Если вы напишете специальный модуль по имени `spammy.module` и включите в него функцию `spammy_user_login()`, которая отправляет пользователю сообщение электронной почты, то будет вызываться и эта функция, и бедный пользователь будет получать ненужные почтовые сообщения при каждом входе.

Реализация хуков в модулях является наиболее распространенным способом подключения к базовой функциональности Drupal.

Совет. Более подробная информация о хуках, которые поддерживает Drupal, находится в электронной документации по адресу <http://api.drupal.org/api/7>, раздел “Components of Drupal” (“Компоненты Drupal”), подраздел “Module system (Drupal hooks)” (“Система модулей (хуки Drupal)”).

Темы

При создании веб-страниц для отправки браузеру возникают две основные проблемы: сбор нужных данных и разметка этих данных для веб. В Drupal за создание кода HTML (или JSON, XML и т.п.), который получит браузер, отвечает уровень тем. В качестве основного механизма шаблонов в Drupal используется технология PHPTemplate, но можно применять и Easy Template System (ETS). При создании новых тем Drupal разработчики обычно пользуются стандартным механизмом шаблонов. Важно помнить, что Drupal поощряет разделение контента и разметки.

В Drupal имеется несколько способов настройки и переопределения внешнего вида веб-сайта. Самый простой путь предусматривает применение каскадных таблиц стилей (CSS) для переопределения встроенных в Drupal классов и идентификаторов. Однако если вы захотите пойти дальше и настроить действительный HTML-вывод, то и это делается легко. Файлы шаблонов Drupal содержат стандартный код HTML и PHP. Кроме того, каждую динамическую часть страницы Drupal (например, список или путь навигации) можно переопределить, просто объявив функцию с соответствующим именем. После этого для создания данной части страницы Drupal будет использовать вашу функцию вместо встроенной.

Ноды

Типы контента в Drupal порождены от единственного базового типа, называемого *нодой* (node). Будь это запись блога, рецепт или даже задача проекта — основная структура данных для них одна и та же. Удобство такого подхода состоит в его легкой расширяемости. Разработчики модуля могут добавлять в ноды такие средства, как рейтинги, комментарии, вложенные файлы, информацию о географическом расположении и т.п., не заботясь о том, является ли тип ноды блогом, рецептом и т.д. Например, администратор может разрешить комментировать блоги, но не рецепты, или разрешить загрузку файлов только для задач проекта.

Ноды также содержат базовый набор свойств, определяющих их поведение, которые наследуются всеми остальными типами контента. Любую ноду можно выдвинуть на первую страницу веб-сайта, опубликовать или отменить публикацию, либо даже сделать так, чтобы в ней работала функция поиска. Благодаря такой унифицированной структуре, в интерфейсе администратора имеется экран пакетного редактирования для работы с нодами.

Поля

Контент в Drupal состоит из отдельных полей. Например, полем является заголовок ноды, а также ее тело. С помощью полей в Drupal можно создавать любой мыслимый тип контента — например, Event (Событие). Обычно событие содержит заголовок, описание (или тело), дату и время начала, продолжительность, местоположение и, возможно, ссылку для регистрации события. Каждый из этих элементов представляет собой поле. В Drupal можно создавать типы контента с помощью полей — программно (за счет создания модуля) или через интерфейс администрирования Drupal, с помощью создания нового типа контента и назначения полей через пользовательский интерфейс. К счастью, интерфейс Field API существенно облегчает создание как простых, так и сложных типов контента почти без программирования.

Блоки

Блок (block) — это информация, которая может быть включена или отключена в определенном месте шаблона веб-сайта. Например, с помощью блока можно отобразить количество активных в данный момент пользователей на сайте. Может существовать блок, содержащий ссылки на самый популярный контент сайта или список ближайших событий. Блоки обычно помещаются в боковую панель, заголовок или нижний колонтитул шаблона. Блоки можно настроить для отображения в нодах определенного типа, только на первой странице или по другим критериям.

Часто блоки используются для представления информации, настроенной для конкретного пользователя. Например, пользовательский блок может содержать ссылки только на те административные области сайта, к которым текущий пользователь имеет права доступа — например, на страницу “My account” (“Моя учетная запись”). Области, где могут появляться блоки (вроде заголовка, нижнего колонтитула, правой или левой боковой панели), определяются темой сайта; размещением и отображением блоков можно управлять через административный веб-интерфейс.

Расположение файлов

Знание структуры каталогов, создаваемой при стандартной установке Drupal, позволит усвоить несколько важных правил. В частности, вы будете знать, где должны размещаться загруженные модули и темы и как использовать различные профили инсталляции Drupal. Стандартная структура каталогов Drupal показана на рис. 1.4.

Ниже приведены пояснения относительно каждого элемента этой структуры.

- Папка `includes` содержит библиотеки общих функций, которые использует Drupal.
- Папка `misc` хранит сценарии JavaScript и различные пиктограммы и изображения, входящие в дистрибутив Drupal.
- Папка `modules` содержит модули ядра, причем каждый модуль хранится в отдельной папке. Лучше ничего не изменять в этой папке (и в любой другой, кроме `profiles` и `sites`). Дополнительные модули должны добавляться в каталог `sites`.
- Папка `profiles` содержит различные профили инсталляции для сайта. Если в этом подкаталоге имеются другие профили, помимо профиля по умолчанию, при первоначальной установке сайта Drupal будет появляться вопрос, какой именно профиль вы хотите использовать. Основное назначение профиля инсталляции — автоматическое включение доступа к определенным модулям ядра и сторонним модулям. Примером может служить профиль электронной торговли, который автоматически настраивает Drupal как платформу для Интернет-магазина.
- Папка `scripts` содержит сценарии, используемые для проверки синтаксиса, очистки кода, запуска Drupal из командной строки, обработки особых случаев с помощью `cron` и запуска тестовых наборов (только в Drupal 7). Они не применяются при выполнении запросов к Drupal, а являются командными сценариями, написанными на языке командной оболочки и Perl.

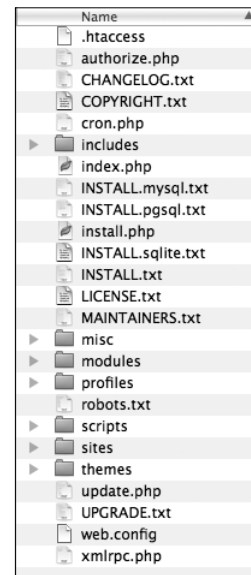


Рис. 1.4. Стандартная структура каталогов Drupal

- Каталог `sites` (рис. 1.5) содержит изменения в Drupal в виде параметров, модулей и тем. При добавлении в Drupal модулей из хранилища дополнительных модулей или в случае создания собственного модуля они помещаются в подкаталог `sites/all/modules`. Благодаря этому все модификации Drupal хранятся в единственной папке. В каталоге `sites` должен быть подкаталог с именем `default`, содержащий файл стандартной конфигурации сайта Drupal — `default.settings.php`. Инсталлятор Drupal модифицирует эти исходные параметры на основе предоставленной вами информации и запишет файл `settings.php` для вашего сайта. Обычно стандартный каталог копируется и переименовывается в соответствии с URL сайта лицом, выполняющим развертывание сайта, поэтому ваш файл настроек может иметь имя вроде `sites/www.example.com/settings.php`.

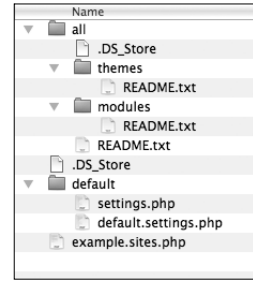


Рис. 1.5. В каталоге `sites` могут храниться все модификации Drupal

- Папка `sites/default/files` создается при базовой инсталляции Drupal по умолчанию. Она необходима для хранения любых файлов, которые загружаются на сайт и впоследствии обслуживаются. Папка создается, например, при применении особого логотипа, разрешении использования посетителями “аватаров” или загрузке других медиафайлов, связанных с новым сайтом. Для этого подкаталога требуются права на чтение и запись со стороны сервера, на котором выполняется Drupal. Инсталлятор Drupal создаст (если сможет) этот подкаталог, и проверит, заданы ли необходимые права доступа. Кроме папки `sites/default/files` может быть создан каталог `sites/default/private` для хранения конфиденциальных файлов, которые не следует разрешать просматривать посетителям сайта без надлежащих полномочий. Для создания этого каталога нужно выбрать пункт меню Configuration⇒File System (Конфигурация⇒Файловая система) и ввести в текстовом поле Private file system path (Путь в приватной файловой системе) имя каталога, где должны храниться конфиденциальные файлы.
- Папка `themes` содержит механизмы шаблонов и стандартные темы для Drupal. Дополнительные темы — загруженные или созданные самостоятельно — не должны попадать сюда; они помещаются в каталог `sites/all/themes`.
- `cron.php` используется для выполнения регулярных задач, таких как сжатие таблиц баз данных и сбор статистики.
- `index.php` — главная точка входа для обслуживания запросов.
- `install.php` — главная точка входа для инсталлятора Drupal.
- `update.php` обновляет схему базы данных после модернизации до новой версии Drupal.
- `xmlrpc.php` принимает запросы XML-RPC и может быть безопасно удален из развертываний, где не нужно принимать такие запросы.
- `robots.txt` — реализация по умолчанию стандарта исключения роботов.
- `authorize.php` — административный сценарий для выполнения авторизованных операций обработки файлов — например, загрузки и установки новой темы или модуля с сайта Drupal.org.

Другие файлы, которые здесь не перечислены — это файлы документации.

Обслуживание запросов

Разумеется, полезно знать в общих чертах, что происходит, когда Drupal получает запрос; поэтому в данном разделе приводится краткий обзор на эту тему. Если вы хотите убедиться в этом самостоятельно, воспользуйтесь хорошим отладчиком и начните со страницы `index.php` — именно на ней Drupal получает основную часть запросов. Последовательность действий, описанная в этом разделе, может показаться слишком сложной для отображения простой веб-страницы, но зато она гибкая.

Роль веб-сервера

Drupal выполняется на веб-сервере, которым обычно является Apache. Если веб-сервер корректно обрабатывает файл `.htaccess` системы Drupal, то он инициализирует некоторые параметры PHP и анализирует URL. Почти все вызовы Drupal выполняются через страницу `index.php`. Например, обращение к `http://example.com/foo/bar` приводит к выполнению следующего процесса.

1. Правило `mod_rewrite` в файле `.htaccess` системы Drupal просматривает входящий URL и разделяет его на базовый URL и путь. В нашем примере указан путь `foo/bar`.
2. Этот путь присваивается параметру URL-запроса по имени `q`.
3. В результате получается URL-адрес `http://example.com/index.php?q=foo/bar`.
4. Drupal трактует `foo/bar` как внутренний путь Drupal, и обработка начинается с `index.php`.

В результате этого процесса Drupal трактует адреса `http://example.com/index.php?q=foo/bar` и `http://example.com/foo/bar` одинаково, поскольку в обоих случаях внутренний путь один и тот же. Это позволяет Drupal использовать URL без экзотических символов внутри. Такие URL-адреса называются чистыми.

При установке Drupal на веб-серверах других типов, таких как Microsoft IIS, чистые URL могут быть получены с помощью модуля ISAPI (Internet Server Application Programming Interface — интерфейс прикладного программирования Интернет-сервера) Windows, такого как ISAPI_Rewrite. Сервер IIS 7 и последующие его версии поддерживают перезапись непосредственно. Если сайт работает под управлением IIS 7 или более поздней версии, необходимо исключить файл `web.config`, который разрешает использование чистых URL и защищает от посторонних глаз файлы, которые лучше им не просматривать, например, `.install`, `.module`, `.test`, `.theme`, `.profile`, `.info` и `.inc`.

Процесс начальной загрузки

Drupal загружает себя при каждом запросе, проходя ряд этапов раскрутки. Эти этапы определены в файле `bootstrap.inc` и выполняются так, как описано в табл. 1.1.

Обработка запроса

Все действия по обработке и сбору данных, необходимых для удовлетворения запроса, выполняются функцией обратного вызова. Например, если получен запрос контента наподобие `http://example.com/q=node/3`, такой URL отображается на функцию `node_page_view()` в модуле `node.module`. При дальнейшей обработке данные для этой ноды извлекаются из базы и помещаются в структуру данных. После этого приходит время темизации.

Таблица 1.1. Процесс загрузки

Этап	Назначение
Конфигурирование	Задаёт глобальные переменные, которые используются в процессе загрузки
СУБД	Инициализирует СУБД и регистрирует функции автозагрузки
Переменные	Загружает системные переменные и все разрешенные модули раскрутки
Сеанс	Инициализирует обработку сеансов
Заголовок страницы	Вызывает метод <code>hook_boot()</code> , инициализирует систему блокировки и посылает стандартные HTTP-заголовки
Язык	Инициализирует все определенные языковые типы
Завершение	Последний этап: к этому моменту система Drupal полностью загружена. Здесь проверяются и исправляются входные данные

Оформление данных

Темизация (theming, или оформление посредством выбранной темы) выполняет преобразование выбранных, измененных или созданных данных в формат HTML (либо XML или другой выходной формат). Drupal использует выбранную администратором тему для придания странице нужного внешнего вида. Затем результирующий вывод отправляется веб-браузеру (или другому HTTP-клиенту).

Резюме

После прочтения этой главы вы должны получить общее представление о том, как работает Drupal и что происходит, когда Drupal обрабатывает запрос. Компоненты, которые образуют процесс обслуживания запрошенной пользователем страницы, будут детально рассмотрены в последующих главах.

ГЛАВА 2

Создание модулей

Модули — базовые блоки, образующие основу Drupal. Кроме того, они представляют собой механизм расширения функциональности изначально поставляемой версии Drupal, которая называется ядром Drupal. Тем, кто не знаком с Drupal, я часто говорю, что модули похожи на элементы конструктора “Лего”. Следование заранее оговоренным правилам позволяет им абсолютно точно сочетаться друг с другом, и такие сочетания модулей позволяют создавать мощные и сложные решения.

Модули Drupal делятся на две общие категории — основные и дополнительные. Основные модули поставляются вместе с Drupal и включают в себя такие модули, как опросы, меню, таксономия, поиск, агрегатор новостей и форумы. Дополнительные модули — это все модули, созданные сообществом, которое расширяет и совершенствует функциональные возможности ядра Drupal. Дополнительных модулей буквально тысячи, они доступны для загрузки по адресу <http://drupal.org/project/modules> и охватывают диапазон от простых модулей для решения одной задачи (вроде вывода текущей даты и времени) до сложных решений наподобие интерфейсов для электронной коммерции.

В настоящей главе вы узнаете, как создать с нуля произвольный модуль. По мере его создания вы будете знакомиться со стандартами, которым должны удовлетворять модули. Лучше ставить реальные цели, и поэтому мы рассмотрим вполне актуальную задачу аннотирования. При просмотре страниц веб-сайта Drupal может возникнуть желание написать что-нибудь о какой-то странице. Для этого можно было бы воспользоваться функцией комментирования Drupal, но комментарии обычно доступны для просмотра всем посетителям сайта или аутентифицированным пользователям, а вот аннотации видны только автору ноды.

Создание файлов

Сначала нужно выбрать имя для модуля. Вполне годится имя “annotate” — и коротко, и понятно. После этого понадобится выбрать место для хранения модуля. Сторонние и разработанные самостоятельно модули хранятся в каталоге `/sites/all/modules`, причем каждый модуль в собственном подкаталоге, имя которого совпадает с именем модуля.

На заметку! Основные модули Drupal хранятся в каталоге `/modules` — это защищает самостоятельно разработанные и сторонние модули от перезаписи или удаления во время модернизации до новой версии.

Можно создать каталог `/sites/all/modules/custom` для хранения всех модулей, созданных с нуля — так вы облегчите жизнь тому, кто будет просматривать ваш сайт

и интересоваться, какие модули загружены с сайта Drupal.org, а какие специально разработаны для данного сайта. Теперь в каталоге `/sites/all/modules/custom` необходимо создать подкаталог `annotate`, который будет содержать все файлы, связанные с модулем аннотирования.

Первым файлом, который мы создадим для нового модуля, будет `annotate.info`. У каждого модуля в Drupal 7 должен быть файл `.info`, имя которого должно совпадать с именем самого модуля. Для модуля аннотации базовая информация, которая необходима для распознавания модуля системой Drupal, выглядит так:

```
name = Annotate
description = "Allows users to annotate nodes."
package = Pro Drupal Development
core = 7.x
files[] = annotate.module
files[] = annotate.install
configure=admin/config/content/annotate/settings
```

Структура этого файла стандартна для всех модулей Drupal 7. Элемент `name` используется для вывода имени модуля на странице `Modules configuration` (Конфигурация модулей). Элемент `description` описывает модуль и также выводится на странице `Modules configuration`. Элемент `package` определяет пакет или группу, с которой связан модуль. На странице `Modules configuration` модули сгруппированы по пакетам. Поле `core` определяет версию Drupal, для которой написан модуль. Элемент `php` указывает версию PHP, необходимую для работы модуля. Наконец, элемент `files` представляет собой массив имен файлов, связанных с модулем. С нашим модулем аннотации связаны файлы `annotate.module` и `annotate.install`.

Кроме вышеперечисленных, можно указать и дополнительные значения. Ниже приведен пример модуля, который требует наличия версии PHP 5.2, а также установленных модулей форума и таксономии.

```
name = Forum confusion
description = Randomly reassigns replies to different discussion threads.
core = 7.x
dependencies[] = forum
dependencies[] = taxonomy
files[] = forumconfusion.module
files[] = forumconfusion.install
package = "Evil Bob's Forum BonusPak"
php = 5.2
```

Теперь все готово для создания самого модуля. Создайте в подкаталоге `sites/all/modules/custom/annotate` файл по имени `annotate.module`. В начало этого файла поместите открывающий дескриптор PHP и дескриптор идентификации CVS, а за ними комментарий:

```
<?php
/**
 * @file
 * Позволяет пользователям добавлять в ноды персональные аннотации.
 *
 * При выводе ноды добавляет текстовое поле, которое позволяет
 * аутентифицированным пользователям вставлять примечания.
 */
```

Обратите внимание на стиль комментария. Он начинается с символов `/**`, продолжается строками со звездочкой с одним предшествующим пробелом (`*`) и завершается

отдельной строкой с символами `*/`. Элемент `@file` означает, что содержимое последующих строк описывает действие файла. Такое однострочное описание используется компонентом автоматического извлечения и форматирования документации `api.module` (см. <http://drupal.org/project/api>). На сайте `Drupal.org` посетите страницу <http://api.drupal.org>. Там находится подробная документация по каждому API-интерфейсу, предоставляемому Drupal. Не поленитесь и просмотрите этот раздел сайта — это бесценный источник для тех, кто разрабатывает или изменяет модули.

После пустой строки добавлено более развернутое описание, предназначенное для программистов, которые будут изучать (и, конечно же, улучшать) наш код. Обратите внимание, что здесь специально не использован закрывающий дескриптор (`?>`) — в PHP они не обязательны и даже могут привести к проблемам с завершающими пробельными символами в файлах (см. <http://drupal.org/coding-standards#phptags>).

На заметку! Почему мы так беспокоимся о структурировании каждой мелочи? Да потому, что если сотни человек со всего мира работают над совместным проектом, стандартизация экономит время. Подробнее о стиле кодирования, принятом в Drupal, см. в разделе “Coding standards” (“Стандарты кодирования”) документа “Developing for Drupal Handbook” (“Справочник разработчика Drupal”), доступного по адресу <http://drupal.org/coding-standards>.

Теперь необходимо определить некоторые параметры, чтобы можно было использовать веб-форму для выбора нод, которые пользователь захочет аннотировать. Для этого понадобится выполнить два шага. Сначала мы определим путь, где будут находиться параметры настройки, а потом создадим форму параметров. Для создания пути необходимо реализовать хук, а именно — `hook_menu`.

Реализация хука

Среда Drupal основана на системе *хуков* (`hook`), которые иногда называют обратными вызовами (`callback`). Во время своей работы Drupal опрашивает модули, хотят ли они сделать что-либо. Например, при загрузке ноды из базы данных перед выводом на странице Drupal просматривает все активные модули и проверяет, реализована ли в них функция `hook_node_load()`. Если это так, Drupal сначала запускает хук этого модуля, а уже потом отображает ноду на странице. Мы рассмотрим работу этого механизма на примере модуля аннотации.

Первым нашим реализованным хуком будет функция `hook_menu()`. С ее помощью мы добавим два элемента в меню администрирования сайта. Мы добавим новый пункт меню “annotate” (аннотировать) в главное меню `admin/config` (администрирования/конфигурирования) и подпункт этого пункта по имени “settings” (параметры), при щелчке на котором будет открываться страница параметров настройки для аннотирования. Значениями элементов меню являются массивы пар “ключ-значение”, которые задают действия Drupal при запросе данного пути. Они будут рассмотрены в главе 4, где описывается система меню и обратных вызовов Drupal. Вызов `hook_menu` мы назовем `annotate_menu` — т.е. заменим `hook` именем нашего модуля. Так принято со всеми хуками: слово `hook` заменяется именем модуля.

Ниже показано, что добавляется в модуль:

```
/**
 * Реализация hook_menu().
 */
function annotate_menu() {
  $items['admin/config/annotate'] = array(
    'title' => 'Node annotation',
```

```

        // Аннотирование ноды
        'description' => 'Adjust node annotation options.',
        // Настройка параметров аннотирования ноды
        'position' => 'right',
        'weight' => -5,
        'page callback' => 'system_admin_menu_block_page',
    // обратный вызов страницы
    'access arguments' => array('administer site configuration'),
    // аргументы доступа
    'file' => 'system.admin.inc',
    'file path' => drupal_get_path('module', 'system'),
);

$items['admin/config/annotate/settings'] = array(
    'title' => 'Annotation settings',
    // Параметры аннотации
    'description' => 'Change how annotations behave.',
    // Изменение поведения аннотаций
    'page callback' => 'drupal_get_form',
    'page arguments' => array('annotate_admin_settings'),
    // аргументы страницы
    'access arguments' => array('administer site configuration'),
    'type' => MENU_NORMAL_ITEM,
    'file' => 'annotate.admin.inc',
);

return $items;
}

```

Не беспокойтесь, если что-то здесь непонятно. Приведенный код означает: “Если пользователь зайдет на страницу <http://example.com/?q=admin/settings/annotate/settings>, необходимо вызвать функцию `drupal_get_form()` и передать ей идентификатор формы `annotate_admin_settings`. Функцию, описывающую эту форму, нужно искать в файле `annotate.admin.inc`. Данный элемент меню должен быть виден только пользователям с правом `administer site configuration` (настройка конфигурации сайта)”. Когда понадобится вывести форму, Drupal запросит определение формы (об этом чуть ниже). После окончания запроса всех модулей для элементов меню будет сформирован список, из которого Drupal выберет нужную функцию для вызова запрошенного пути.

На заметку! Если вам интересно познакомиться с функцией, лежащей в основе механизма хуков, рассмотрите функцию `module_invoke_all()` в каталоге `includes/module.inc` (см. http://api.drupal.org/api/function/module_invoke_all/7).

Теперь вы должны понимать, почему мы назвали функцию `hook_menu()` или хуком меню.

Совет. Механизм хуков Drupal позволяет модифицировать почти все аспекты программного обеспечения. Полный список поддерживаемых хуков и способов их применения приведен в документации по Drupal API (<http://api.drupal.org/api/group/hooks/7>).

Добавление параметров для конкретного модуля

В Drupal поддерживается множество различных типов нод (в пользовательском интерфейсе они называются *типами контента* (content type)), такие как статьи и базо-

вые страницы. Мы ограничим использование аннотаций только некоторыми типами нод. Для этого понадобится создать страницу, где можно сообщить модулю о том, какие типы контента мы хотим аннотировать. На этой странице будет представлен набор флажков — по одному для каждого существующего типа контента. Это позволит конечному пользователю указать, какие типы контента могут аннотироваться, сбросив или установив соответствующие флажки (рис. 2.1). Такая страница является административной, а это значит, что ее код необходимо загружать и синтаксически анализировать только при необходимости. Поэтому мы поместим этот код в отдельный файл, а не в `annotate.module`, который будет загружаться и выполняться при каждом веб-запросе. Поскольку Drupal было указано, что искать форму параметров нужно в файле `annotate.admin.inc`, мы создадим файл `sites/all/modules/annotate/annotate.admin.inc` и поместим в него следующий код:

```
<?php

/**
 * @file
 * Обратные вызовы административной страницы для модуля аннотирования.
 */

/**
 * Создание формы. Настройка аннотирования.
 */
* @ingroup forms
* @see system_settings_form().
*/
function annotate_admin_settings() {
  // Получение массива типов нод: ключи — это внутренние имена,
  // а значения — "дружественные имена". Например:
  // array('page' => 'Basic Page', 'article' => 'Articles')

  $types = node_type_get_types();
  foreach($types as $node_type) {
    $options[$node_type->type] = $node_type->name;
  }
  $form['annotate_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Users may annotate these content types'),
    // Пользователи могут аннотировать эти типы контента
    '#options' => $options,
    '#default_value' => variable_get('annotate_node_types', array('page')),
    '#description' => t('A text field will be available on these content
types to make author-specific notes.'),
    // Для этих типов контента будут доступны текстовые
    // поля, предназначенные для авторских заметок
  );

  return system_settings_form($form, TRUE);
}
```

Формы в Drupal представляются в виде вложенной древовидной структуры — т.е. массива массивов. Эта структура описывает, каким образом механизм отображения форм Drupal должен представлять эту форму. Для наглядности лучше размещать каждый элемент массива в отдельной строке. Средства формы отмечаются знаком # и играют роль ключей массива. Вначале мы объявим, что тип элемента формы должен быть `checkboxes`; это означает, что на основании ассоциативного массива будет создано несколько флажков. Этот массив уже содержится в переменной `$options`.

Содержимое переменной `$options` присваивается выходным данным функции `node_type_get_types()`, которая возвращает массив объектов. Эти выходные данные должны выглядеть примерно так:

```
[article] => stdClass Object (
    [type] => article
    [name] => Article
    [base] => node_content
    [description] => Use articles for time-sensitive content like news,
    press releases or blog posts.
    [help] =>
    [has_title] => 1
    [title_label] => Title
    [has_body] => 1
    [body_label] => Body
    [custom] => 1
    [modified] => 1
    [locked] => 0
    [orig_type] => article
)
```

Ключами массива объектов являются внутренние имена Drupal для типов нод, а дружественные имена (те, которые выводятся пользователю) содержатся в атрибуте `name` объекта.

Согласно требованиям API-интерфейса форм Drupal, `#options` должно быть массивом пар `ключ => значение`. Это позволит циклу `foreach` использовать атрибут `type` для создания ключа и атрибут `name` для создания значения нового массива, который здесь называется `$options`. На основе значений из массива `$options` в нашей веб-форме Drupal сгенерирует флажки для типов нод `Basic page` и `article`, а также для всех других типов контента, доступных на сайте.

Заголовок элементу формы назначается определением значения свойства `#title`.

На заметку! Любой текст, который выводится пользователю (вроде свойств `#title` и `#description` из нашего поля формы), необходимо указывать внутри функции `t()` — это функция Drupal, предназначенная для облегчения перевода строк. Преобразование всего текста через эту функцию строкового перевода значительно облегчает локализацию модуля на различные языки. Для пункта меню здесь это не сделано потому, что пункты меню переводятся автоматически.

Следующая директива — `#default_value` — задает стандартное значение для данного элемента формы. Поскольку `checkboxes` является множественным элементом формы (т.е. флажков несколько), значением `#default_value` будет массив.

Это значение `#default_value` стоит рассмотреть более внимательно:

```
variable_get('annotate_node_types', array('page'))
```

Drupal позволяет программистам сохранять и выбирать любые значения с помощью специальной пары функций: `variable_get()` и `variable_set()`. Значения хранятся в базе данных `variables` и доступны в любой момент при обработке запроса. А поскольку эти переменные выбираются из базы данных при каждом запросе, не следует хранить таким способом большие объемы данных. Обратите внимание: функции `variable_get()` передается (для выборки) ключ, который описывает значение, и значение по умолчанию. В рассматриваемом случае значением по умолчанию является массив с типами нод, которые допускают аннотирование. По умолчанию мы разрешим аннотирование для типов контента базовых страниц (`Basic Page`).

Совет. При использовании функции `system_settings_form()` имя элемента формы (в данном случае — `annotate_node_types`) должно совпадать с именем ключа, передаваемым в `variable_get()`.

Здесь добавлено описание, которое немного поможет администратору сайта понять, что следует поместить в данное поле. О формах речь пойдет в главе 11.

Теперь добавим код для обработки добавления и удаления поля аннотации в типах контента. Если администратор сайта отмечает тип контента, к этому типу контента должно быть добавлено поле для аннотации. Если администратор сайта решит убрать поле аннотации из типа контента, это поле должно быть удалено. Для определения поля и связывания его с типом контента мы используем интерфейс Field API системы Drupal. Этот API-интерфейс обрабатывает все действия, связанные с настройкой поля, включая создание таблицы в базе данных Drupal для хранения значений, отправленных авторами контента, создание элемента формы, который будет использоваться для сбора информации, вводимой автором, и связывание поля с типом контента и вывод этого поля на форме редактирования ноды при отображении ноды на странице. Программный интерфейс Field API будет рассмотрен в главе 4.

Вначале понадобится создать подпрограмму проверки, которая будет вызываться при отправке формы администратором сайта. В этой подпрограмме модуль проверит, установлен или сброшен флажок для типа контента. Если он сброшен, будет выполнена проверка, что тип контента не содержит связанного с ним поля аннотации. Если содержит, это означает, что администратор сайта желает удалить такое поле из данного типа контента, и нужно удалить аннотации, которые уже занесены в базу данных. Если флажок установлен, модуль проверяет, существует ли поле в данном типе контента, и если нет, модуль добавляет такое поле к типу контента.

```
/**
 * annotate_admin_settings_validate
 */
function annotate_admin_settings_submit($form, $form_state) {
  // Цикл по всем флажкам типов контента, имеющихся на форме.
  foreach ($form_state['values']['annotate_node_types'] as $key => $value) {
    // Если флажок для типа контента сброшен, проверяем,
    // имеется ли у данного типа контента связанное с ним поле аннотации,
    // с помощью функции field_info_instance. Если имеется,
    // нужно удалить поле аннотации, т.к. флажок сброшен.
    if (!$value) {
      $instance = field_info_instance('node', 'annotation', $key);
      if (!empty($instance)) {
        field_delete_instance($instance);
        watchdog("Annotation", "Deleted annotation field from content type: ".$key);
      } // Конец варианта, когда аннотация существует, и ее необходимо удалить.
    } else {
      // Если флажок для типа контента установлен, проверяем, связано ли с данным
      // типом контента поле. Если нет, добавим в тип контента поле аннотации.
      $instance = field_info_instance('node', 'annotation', $key);
      if (empty($instance)) {
        $instance = array(
          'field_name' => 'annotation',
          'entity_type' => 'node',
          'bundle' => $key,
          'label' => t('Annotation'),
        );
      }
    }
  }
}
```

```

        'widget_type' => 'text_textarea_with_summary',
        'settings' => array('display_summary' => TRUE),
        'display' => array(
            'default' => array(
                'type' => 'text_default',
            ),
            'teaser' => array(
                'type' => 'text_summary_or_trimmed',
            ),
        ),
    );
};

$instance = field_create_instance($instance);

watchdog("Annotation", "Added annotation field to content type: ".$key);
} // Конец варианта, когда поля аннотации нет, и оно добавляется.
} // Конец оператора if.

} // Конец оператора foreach.
}

```

Теперь нужно создать файл `.install` для нашего модуля. Этот файл содержит одну или несколько функций, которые вызываются при инсталляции или удалении модуля. В нашем случае при инсталляции модуля должно быть создано поле аннотации, чтобы администраторы сайта могли связывать его с типами контента. При удалении модуля необходимо удалить поле аннотации из всех типов контента и удалить поле с его содержанием из базы данных Drupal. Для этого в каталоге модуля аннотирования понадобится создать новый файл по имени `annotate.install`.

Первая вызываемая функция — `hook_install()`. В соответствии со стандартами именования функций хуков в Drupal мы назовем ее `annotate_install()`, заменив слово `hook` именем модуля. В функции `hook_install` будет выполняться проверка существования поля с помощью Field API, и если оно не существует, будет создаваться поле аннотации.

```

<?php
/**
 * Реализация hook_install()
 */

function annotate_install() {
    // Проверка, существует ли поле аннотации.
    $field = field_info_field('annotation');

    // Если поле аннотации не существует, оно создается.
    if (empty($field)) {
        $field = array(
            'field_name' => 'annotation',
            'type' => 'text_with_summary',
            'entity_types' => array('node'),
            'translatable' => TRUE,
        );
        $field = field_create_field($field);
    }
}

```

После этого нужно создать функцию удаления (деинсталляции) с помощью `hook_uninstall`. Мы создадим функцию с именем `annotate_uninstall` и воспользуемся функцией отслеживания `watchdog` для записи сообщения администратору сайта о том, что модуль деинсталлирован. Затем применим API-функцию `node_get_types()` для сбора списка всех типов контента, которые имеются на сайте, с последующим просмотром этого списка типов, чтобы определить, существует ли поле аннотации для данного типа контента. Если да, оно удаляется. После этого можно удалить само поле аннотации.

```
/**
 * Реализация hook_uninstall()
 */
function annotate_uninstall() {
  watchdog("Annotate Module", "Uninstalling module and deleting fields");

  $types = node_type_get_types();
  foreach($types as $type) {
    delete_annotation($type);
  }

  $field = field_info_field('annotation');
  if ($field) {
    field_delete_field('annotation');
  }
}

function delete_annotation($type) {
  $instance = field_info_instance('node', 'annotation', $type->type);
  if ($instance) {
    field_delete_instance($instance);
  }
}
```

Последним шагом процесса должно быть включение в файл `.module` проверки, является ли лицо, просматривающее ноду, автором этой ноды. Если это не автор, то поле аннотирования нужно скрыть. Мы применим простой подход с помощью функции `hook_node_load()` — хука, который вызывается при загрузке ноды. Внутри функции `hook_node_load()` будет выполняться проверка, является ли пользователь, просматривающий ноду, его автором. Если это не автор, аннотация скрывается с помощью функции сброса.

```
/**
 * Реализация hook_node_load()
 */
function annotate_node_load($nodes, $types) {
  global $user;

  // Проверка, является ли лицо, просматривающее ноду, автором.
  // Если нет, аннотация скрывается.
  foreach ($nodes as $node) {
    if ($user->uid != $node->uid) {
      unset($node->annotation);
    }
  }
}
```

Сохраните созданные файлы (.info, .install, .admin.inc, .module) и щелкните на пункте Modules (Модули) в меню администрирования в верхней части страницы. Наш модуль должен появиться в группе Pro Drupal Development (если не появился, перепроверьте синтаксис файлов `annotate.info` и `annotate.module`; они должны находиться в каталоге `sites/all/modules/custom`). Активируйте новый модуль.

После активации модуля перейдите по адресу `admin/config/annotate/settings` — там должна появиться форма настройки для `annotate.module` (рис. 2.1).

Рис. 2.1. Сгенерированная форма настройки для `annotate.module`

Написав лишь несколько строк кода, мы получили функциональную форму настройки для нашего модуля, которая автоматически сохраняет и запоминает параметры! Это дает некоторое представление о мощи, заложенной в Drupal.

Давайте протестируем процесс. Для этого сначала разрешим аннотирование для всех типов контента. Установите все флажки на странице настройки параметров и щелкните на кнопке `Save configuration` (Сохранить конфигурацию). Потом создайте новую ноду базовой страницы и прокрутите ее вниз, пока не увидите поле аннотирования (рис. 2.2).

Рис. 2.2. Форма аннотирования на веб-странице Drupal

Создайте новую ноду, введя значения в поле заголовка, тела и аннотации. После этого щелкните на кнопке сохранения — результат должен быть похож на показанный на рис. 2.3.

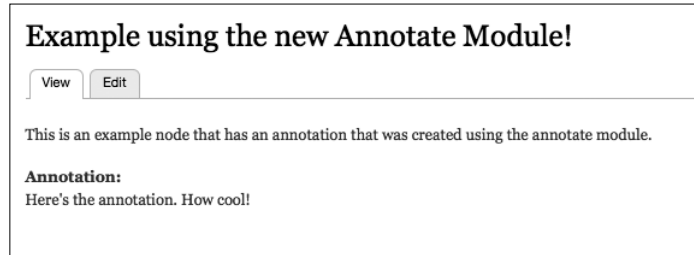


Рис. 2.3. Аннотированная нода

Здесь, даже неявно, не выполнялось ни одной операции с базами данных — непонятно, где Drupal сохраняет и откуда выбирает значения для поля аннотации? Все скрытые операции по созданию таблицы для хранения значения, а также сохранение и выборку значений при сохранении и загрузке ноды выполняет интерфейс Field API. При вызове функции `field_create_field()` из Field API создает таблицу в базе данных Drupal с помощью стандартного соглашения об именовании `field_data_<имя_поля>`. В случае нашего поля аннотирования таблица называется `field_data_annotations`. Более подробно Field API рассматривается в главе 4.

Определение собственного раздела администрирования

В Drupal имеется несколько категорий административных параметров — например, для управления контентом и управления пользователями — которые выводятся на странице Configuration (Конфигурация). Если для какого-то модуля нужна особая категория, ее нетрудно создать. В нашем примере мы создадим новую категорию с названием Node annotation (Аннотирование ноды). Для этого потребуется создать хук для меню модуля и определить новую категорию:

```
/**
 * Реализация hook_menu().
 */
function annotate_menu() {
  $items['admin/config/annotate'] = array(
    'title' => 'Node annotation',
    'description' => 'Adjust node annotation options.',
    'position' => 'right',
    'weight' => -5,
    'page callback' => 'system_admin_menu_block_page',
    'access arguments' => array('administer site configuration'),
    'file' => 'system.admin.inc',
    'file path' => drupal_get_path('module', 'system'),
  );

  $items['admin/config/annotate/settings'] = array(
    'title' => 'Annotation settings',
    'description' => 'Change how annotations behave.',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('annotate_admin_settings'),
    'access arguments' => array('administer site configuration'),
  );
}
```

```

    'type' => MENU_NORMAL_ITEM,
    'file' => 'annotate.admin.inc',
  );
  return $items;
}

```

Результат этого изменения кода, а именно — новая категория на странице Configuration со ссылкой на параметры нашего модуля, показан на рис. 2.4.

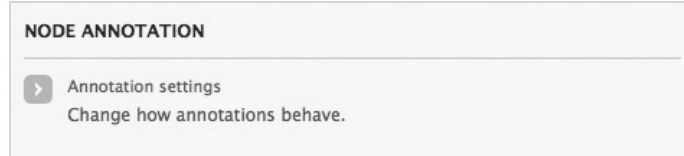


Рис. 2.4. Ссылка на параметры модуля аннотации теперь выделены в отдельную категорию

Если вы выполняете все примеры, то чтобы ссылка появилась, понадобится очистить кэш меню. Для этого нужно либо удалить данные из таблицы `cache_menu`, либо щелкнуть на ссылке **Rebuild menus** (Пересоздать меню) в модуле разработки Drupal (`devel.module`), либо с помощью кнопки **Clear cached data** (Очистить кэшированные данные), зайдя на страницу Configuration (Настройка) и щелкнув на ссылке **Performance** (Производительность).

Совет. Модуль разработки (<http://drupal.org/project/devel>) написан специально для поддержки разработки для Drupal. Он предоставляет быстрый доступ ко многим функциям разработки: очистке кэша, просмотру переменных, отслеживанию запросов и т.п. Это необходимый инструмент для серьезных проектов.

Мы создали новую категорию в два этапа. Сначала мы добавили пункт меню, который описывает заголовок категории. Этот пункт имеет уникальный путь (`admin/config/annotate`). Его следует поместить в правый столбец с весом `-5`, т.к. тогда он будет расположен как раз над категорией **Web Services** (Веб-службы), что удобнее всего для снимка экрана, показанного на рис. 2.3.

На втором этапе нужно было указать системе Drupal разместить ссылку на параметры аннотирования в категории **Node annotation** (Аннотирование ноды). Для этого мы изменили путь к первоначальному пункту меню `admin/settings/annotate` на `admin/config/annotate/settings`. Когда среда Drupal перестраивает дерево меню, она просматривает пути для создания отношений между родительскими и дочерними пунктами и определяет, что поскольку `admin/config/annotate/settings` является потомком `admin/config/annotate`, он должен отображаться соответственно.

Drupal загружает только те файлы, которые необходимы для выполнения запроса. Это экономит память. Поскольку обратный вызов страницы указывает на функцию за пределами области действия модуля (функция `system_admin_menu_block_page()` в модуле `system.module`), необходимо указать Drupal загрузить файл `modules/system/system.admin.inc`, а не `sites/all/modules/custom/annotate/system.admin.inc`.

Конечно, это не очень естественный пример, и в реальности необходимо иметь веские причины для создания новой категории, чтобы запутывать администратора (обычно себя!) обилием категорий.

Вывод формы параметров пользователю

В модуле аннотирования администратору предоставлена возможность указать, какие типы нод должны поддерживать аннотирование (см. рис. 2.1). Рассмотрим этот механизм.

Когда администратору сайта потребуется изменить параметры модуля аннотирования, нужно будет вывести форму, в которой администратор сможет выбрать необходимые параметры. В нашем пункте меню обратный вызов страницы указывает на функцию `drupal_get_form()`, а аргументы страницы являются массивом, содержащим `annotate_admin_settings`. Это означает, что при заходе на страницу `http://example.com/?q=admin/config/annotate/settings` будет выполнен вызов `drupal_get_form('annotate_admin_settings')`, который и указывает Drupal создать форму, определенную функцией `annotate_admin_settings()`.

Рассмотрим функцию, определяющую форму — она определяет флажок для типов нод (см. рис. 2.1) и добавляет два дополнительных параметра. Эта функция находится в каталоге `sites/all/modules/custom/annotate/annotate.admin.inc`:

```
/**
 * Создатель формы. Настройка аннотаций.
 *
 * @ingroup forms
 * @see system_settings_form().
 */
function annotate_admin_settings() {
  // Получение массива типов нод с внутренними именами в качестве ключей
  // и "дружественными именами" в качестве значений. Например:
  // array('page' => 'Basic Page', 'article' => 'Articles')
  $types = node_type_get_types();
  foreach($types as $node_type) {
    $options[$node_type->type] = $node_type->name;
  }

  $form['annotate_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Users may annotate these content types'),
    // Пользователи могут аннотировать эти типы контента
    '#options' => $options,
    '#default_value' => variable_get('annotate_node_types', array('page')),
    '#description' => t('A text field will be available on these content types
    to make user-specific notes.'),
    // У этих типов контента пользователю будет доступно
    // текстовое поле для внесения личных примечаний.
  );

  $form['annotate_deletion'] = array(
    '#type' => 'radios',
    '#title' => t('Annotations will be deleted'),
    // Аннотации будут удалены
    '#description' => t('Select a method for deleting annotations.'),
    // Выберите метод для удаления аннотаций
    '#options' => array(
      t('Never'), // Никогда
      t('Randomly'), // Случайным образом
      t('After 30 days') // Спустя 30 дней
    ),
    '#default_value' => variable_get('annotate_deletion', 0)
  );
}
```



```

// По умолчанию Never
);
$form['annotate_limit_per_node'] = array(
  '#type' => 'textfield',
  '#title' => t('Annotations per node'),
  // Аннотаций в ноде
  '#description' => t('Enter the maximum number of annotations allowed per
    node (0 for no limit).'),
  // Введите максимальное количество аннотаций, допустимое для ноды
  // (0 — неограниченное).
  '#default_value' => variable_get('annotate_limit_per_node', 1),
  '#size' => 3
);
return system_settings_form($form, TRUE);
}

```

Здесь добавлен переключатель для указания, когда следует удалять аннотации, и текстовое поле для ввода количества аннотаций, разрешенных для ноды (реализация этих усовершенствований оставлена в качестве самостоятельного упражнения). Вместо управления обработкой нашей формы выполняется вызов `system_settings_form()`, который указывает системному модулю добавить на форму несколько кнопок и взять на себя управление проверкой и отправкой введенных данных. Внешний вид формы с параметрами показан на рис. 2.5.

Рис. 2.5. Усовершенствованная форма с параметрами, использующая флажки, переключатели и текстовое поле

Проверка параметров, введенных пользователем

Если функция `system_settings_form()` выполняет за нас сохранение значений из формы, то каким образом проверить, является ли текст, введенный в поле `Annotations per node` (Аннотаций на ноду), действительно числом? Для этого достаточно добавить проверку, является ли значение числовым, в существующую функцию проверки

(`annotate_admin_settings_validate($form, $form_state)`) в файле `sites/all/modules/custom/annotate/annotate.admin.inc` и использовать ее для генерации ошибки, если что-то не подойдет:

```
$limit = $form_state['values']['annotate_limit_per_node'];
if (!is_numeric($limit)) {
    form_set_error('annotate_limit_per_node', t('Please enter a number.'));
}
...
```

Когда впоследствии Drupal обрабатывает форму, она выполняет для проверки данных обратный вызов `annotate_admin_settings_validate()`. Если определено, что введенное значение неверно, генерируется ошибка, связанная с соответствующим полем, и это отображается на экране в виде предупреждения и выделения поля с ошибкой.

А как Drupal узнает, что нужно вызывать нашу функцию? Для этого мы назвали ее специальным образом: имя функции определения формы (`annotate_admin_settings`) с добавлением `_validate`. Полная информация о том, как Drupal определяет вызываемую функцию проверки формы, приведена в главе 10.

Хранение параметров

В приведенном ниже разделе выполняется изменение значений и щелчок на кнопке `Save configuration` (Сохранить конфигурацию).

Использование таблицы `variables`

Вначале рассмотрим поле `Annotations per node` (Аннотаций на ноду). В его ключ `#default_value` заносится значение `variable_get('annotate_limit_per_node', 1)`.

В базе данных Drupal имеется таблица `variables`; пары “ключ/значение” могут сохраняться в ней с помощью функции `variable_set($key, $value)`, а выбираться — с помощью функции `variable_get($key, $default)`. Так что вызов из предыдущего абзаца означает: “Установить для поля `Annotations per node` в качестве стандартного значение переменной `annotate_limit_per_node`, хранящееся в таблице базы данных `variables`, а если никакого значения не найдено, использовать 1”.

Внимание! Чтобы сохранять и выбирать параметры из таблицы `variables` без возникновения конфликтов пространств имен, всегда назначайте одинаковые имена элементу формы и ключу переменной (в предыдущем примере это `annotate_limit_per_node`). Имена элемента формы и ключа переменной должны образовываться из имени модуля с добавлением описательного имени.

Поле `Annotations will be deleted` (Аннотации будут удалены) несколько сложнее, т.к. оно представляет собой переключатель. Параметр `#options` для него выглядит так:

```
'#options' => array(
    t('Never'),           // Никогда
    t('Randomly'),       // Случайным образом
    t('After 30 days')    // Спустя 30 дней
)
```

Если передать PHP массив без ключей, ему неявно присваиваются числовые ключи, так что внутреннее представление массива будет выглядеть следующим образом:

```
'#options' => array(
    [0] => t('Never'),     // Никогда
    [1] => t('Randomly'),  // Случайным образом
)
```

```
[2] => t('After 30 days')    // Спустя 30 дней
)
```

Указание стандартного значения для этого поля выполняется так:

```
'#default_value' => variable_get('annotate_deletion', 0)
// По умолчанию – Never
```

что означает выбор по умолчанию элемента 0 в массиве, т.е. `t('Never')`.

Извлечение сохраненных значений с помощью функции `variable_get()`

Для извлечения сохраненных параметров должна использоваться функция `variable_get()`:

```
// Извлечение сохраненного параметра – максимального количества аннотаций в ноде.
$max = variable_get('annotate_limit_per_node', 1);
```

Обратите внимание, что для `variable_get()` также применяется значение по умолчанию — на тот случай, если сохраненного значения нет (возможно, администратор еще не заходил на страницу настроек).

Дальнейшие шаги

Естественно, мы поделимся нашим модулем с сообществом пользователей открытого исходного кода. Поэтому необходимо создать файл `README.txt` и сохранить его в каталоге аннотирования вместе с файлами `annotate.info`, `annotate.module` и `annotate.install`. Файл `README.txt` обычно содержит информацию о том, кто написал данный модуль и как его устанавливать. Информация о лицензировании не требуется, т.к. для всех модулей, загруженных на сайт `drupal.org`, подразумевается лицензия GPL, и упаковочный сценарий на `drupal.org` автоматически добавляет файл `LICENSE.txt`. После этого его можно загрузить в хранилище дополнительных файлов на `drupal.org` и создать страницу проекта для отслеживания отзывов от других членов сообщества.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Создавать с нуля модуль Drupal.
- Знать, как внедриться в выполнение кода Drupal.
- Сохранять и извлекать параметры, относящиеся к модулю.
- Создавать и обрабатывать простые формы с помощью интерфейса Forms API системы Drupal.
- Создавать новые административные категории на главной странице администрирования Drupal.
- Определять формы для администратора сайта, чтобы он мог задавать параметры с помощью флажков, текстовых полей ввода и переключателей.
- Выполнять проверку параметров и выводить сообщения об ошибке, если проверка закончилась неудачей.
- Знать, как Drupal хранит и извлекает значения с помощью встроенной системы хранения переменных.

ГЛАВА 3

Хуки, действия и триггеры

Обычно при работе с Drupal требуется каким-то образом реагировать на определенные события. Например, администратору сайта может понадобиться почтовое сообщение об отправке посетителем текста на сайт. Или следует заблокировать пользователя, если он употребляет недопустимые слова. В данной главе описано, как реагировать на события Drupal с помощью собственного кода.

События и триггеры

Drupal при своей работе проходит через последовательность событий. Эти внутренние события — как раз те моменты, когда модулям разрешено взаимодействовать с процессом обработки Drupal. Некоторые события Drupal приведены в табл. 3.1.

Таблица 3.1. Примеры событий Drupal

Событие	Тип
Создание ноды	node
Удаление ноды	node
Просмотр ноды	node
Создание пользовательской учетной записи	user
Изменение пользовательского профиля	user
Вход пользователя	
Выход пользователя	

Разработчики Drupal называют эти внутренние события *хуками* (hook), т.к. при возникновении одного из таких событий Drupal позволяет модулям в этой точке “вмешиваться” (hook into) в путь выполнения. Некоторые хуки уже встречались в предыдущих главах. Обычно при разработке модулей требуется решать, на какие события Drupal необходимо реагировать — т.е. какие хуки нужно реализовать в модуле.

Предположим, что вы недавно создали веб-сайт, который обслуживался стоящим в кладовке компьютером. Потом сайт набрал популярность, и вы планируете продать его большой корпорации и получить кучу денег. Между тем, надо каким-то образом получать уведомления о входе пользователей на сайт. Допустим, вы решили, что в это вре-

моя компьютер должен издавать звуковые сигналы. Но вашей кошке не нравятся такие звуки, когда она спит, и вы решили вместо гудков просто создавать запись в журнале. Вы быстро пишете .info-файл и помещаете его в sites/all/modules/custom/beep/beep.info:

```
name = Beep
description = Simulates a system beep.
package = Pro Drupal Development
core = 7.x
files[] = beep.module
```

После этого необходимо написать модуль sites/all/modules/custom/beep/beep.module:

```
<?php
/**
 * @file
 * Имитация звукового сигнала.
 */

function beep_beep() {
  watchdog('beep', 'Beep!');
}
```

Данный код записывает в журнал Drupal сообщение Beep! — пока этого хватит. А теперь нужно указать среде Drupal, чтобы она имитировала гудок при каждом входе пользователя. Это можно сделать, реализовав в нашем модуле функцию hook_user() и организовав перехват операции login:

```
/**
 * Реализация hook_user().
 */
function beep_user($op, &$edit, &$account, $category = NULL) {
  if ($op == 'login') {
    beep_beep();
  }
}
```

Оказывается, все совсем не трудно. А как насчет сигналов при добавлении нового контента? Это можно сделать, реализовав в модуле функцию hook_node_insert() и перехватив операцию insert:

```
/**
 * Реализация hook_node_insert().
 */
function hook_node_insert($node) {
  beep_beep();
}
```

А если сигнал нужен при добавлении комментария? Тогда можно реализовать функцию hook_comment_insert() и перехватывать операцию insert. Только давайте остановимся на минутку и подумаем. Мы все время делаем одно и то же. И лучше было бы иметь графический пользовательский интерфейс, который позволял бы связать действие (гудок) с любым хуком и любой операцией. Как раз для этого и предназначен встроенный в Drupal модуль триггера. Он позволяет связать действие с заданным событием. В коде событие определяется как уникальное сочетание хука и операции — например, перехват пользователя и операция входа или перехват ноды и операция вставки. При выполнении таких операций модуль trigger.module позволяет запустить выполнение действия.

Во избежание путаницы давайте уточним терминологию.

- *Событие* (event). Обычно в программировании этот термин понимается как сообщение, отправленное от одного компонента системы другим компонентам.
- *Хук* (hook). Программный прием, который применяется в Drupal и позволяет модулям вмешиваться в поток управления. Для каждой операции имеются уникальные хуки, которые выполняются для “перехватываемого” объекта (например, `hook_node_insert`).
- *Триггер* (trigger). Специальное сочетание хука и операции, с которым можно связать одно или более действий. Например, действие звукового сигнала можно связать с операцией `login` хука пользователя.

Действия

Действие (action) — это что-то, выполняемое Drupal. Вот несколько примеров:

- Продвижение ноды на главную страницу
- Изменение состояния ноды с неопубликованного на опубликованное
- Удаление пользователя
- Отправка почтового сообщения

В каждом из этих случаев имеется четко определенная задача. Программисты могут отметить сходство приведенного списка с функциями PHP. Например, почтовое сообщение можно послать с помощью функции `drupal_mail()` из `includes/mail.inc`. Действия похожи на функции потому, что действия *и являются* функциями. Эти функции Drupal может анализировать и слабо связывать с событиями (об этом речь пойдет чуть ниже). А пока рассмотрим модуль триггера.

Пользовательский интерфейс триггера

Выберите пункт меню Modules (Модули) в верхней части страницы и на странице Modules активируйте модуль триггера. Щелкните на ссылке Structure (Структура) в верхней части страницы, а затем на странице Structure щелкните на ссылке Triggers (Триггеры). Появится интерфейс, похожий на показанный на рис. 3.1.

Обратите внимание на вкладки в верхней части. Они соответствуют хукам Drupal! На рис. 3.1 мы видим операции для хука ноды. Им всем назначены удобные имена, например, операция удаления хука ноды помечена текстом Trigger: After deleting content (Триггер: после удаления контента). Так что каждая операция хука показана с возможностью назначения действия, такого как Publish Content (Опубликовать контент), при выполнении этой операции. Все возможные действия приведены в раскрывающемся списке Choose an action (Выберите действие).

На заметку! Не для всех триггеров доступны все действия, т.к. некоторые действия в определенных контекстах не имеют смысла. Например, невозможно выполнить действие “Переместить заметку на главную страницу” с триггером “После удаления контента”. В зависимости от конкретной установки, некоторые триггеры могут выдать сообщение No actions available for this trigger (Для данного триггера нет доступных действий).

Имена некоторых триггеров с соответствующими им хуками и операциями приведены в табл. 3.2.

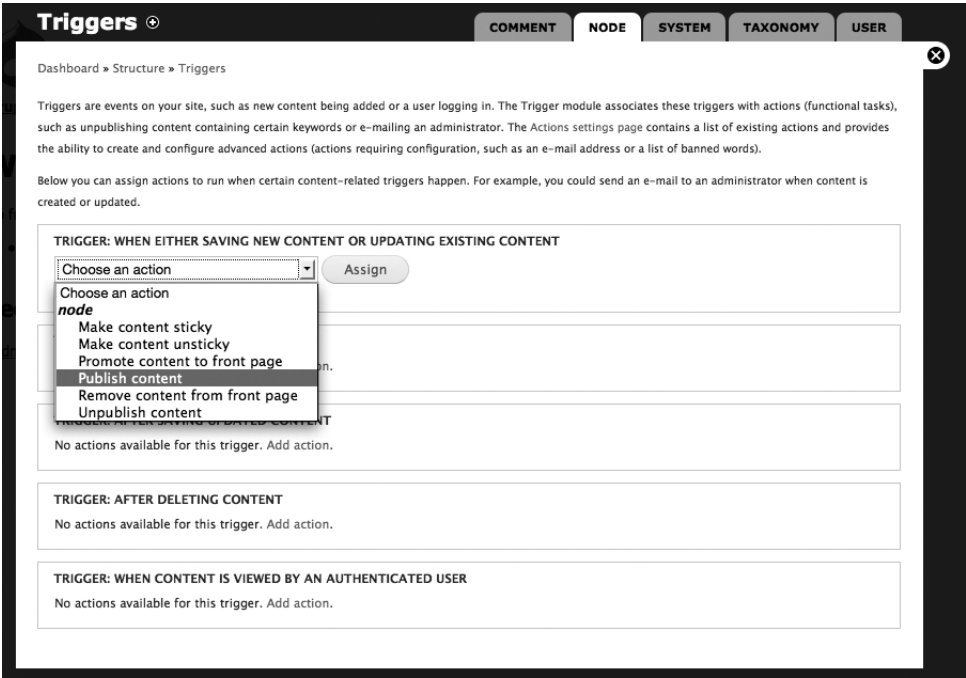


Рис. 3.1. Интерфейс назначения триггера

Таблица 3.2. Взаимосвязь хуков и триггеров в Drupal 7

Хук	Имя триггера
<code>comment_insert</code>	After saving a new comment (После сохранения нового комментария)
<code>comment_update</code>	After saving an updated comment (После сохранения измененного комментария)
<code>comment_delete</code>	After deleting a comment (После удаления комментария)
<code>comment_view</code>	When a comment is being viewed by an authenticated user (При просмотре комментария аутентифицированным пользователем)
<code>cron</code>	When <code>cron</code> runs (При запуске <code>cron</code>)
<code>node_presave</code>	When either saving a new post or updating an existing post (При сохранении нового сообщения или изменении существующего сообщения)
<code>node_insert</code>	After saving a new post (После сохранения нового сообщения)
<code>node_update</code>	After saving an updated post (После сохранения измененного сообщения)
<code>node_delete</code>	After deleting a post (После удаления сообщения)
<code>node_view</code>	When content is viewed by an authenticated user (При просмотре контента аутентифицированным пользователем)
<code>taxonomy_term_insert</code>	After saving a new term to the database (После сохранения в базе данных нового термина)

Хук	Имя триггера
<code>taxonomy_term_update</code>	After saving an updated term to the database (После сохранения в базе данных измененного термина)
<code>taxonomy_term_delete</code>	After deleting a term (После удаления термина)
<code>user_insert</code>	After a user account has been created (После создания пользовательской учетной записи)
<code>user_update</code>	After a user's profile has been updated (После изменения пользовательской учетной записи)
<code>user_delete</code>	After a user has been deleted (После удаления пользователя)
<code>user_login</code>	After a user has logged in (После входа пользователя)
<code>user_logout</code>	After a user has logged out (После выхода пользователя)
<code>user_view</code>	When a user's profile is being viewed (При просмотре пользовательского профиля)

Первое действие

Что же понадобится предпринять, чтобы наша функция выдачи звукового сигнала стала полноправным действием? Для этого необходимо выполнить два шага.

1. Сообщить Drupal, какие триггеры должно поддерживать действие.
2. Создать саму функцию действия.

Первый шаг выполняется за счет реализации функции `hook_action_info()`. В модуле звукового сигнала она должна выглядеть так:

```
/**
 * Реализация hook_action_info().
 */
function beep_action_info() {
  return array(
    'beep_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep annoyingly'), // Противно гудит
      'configurable' => FALSE,
      'triggers' => array('node_view', 'node_insert', 'node_update', 'node_delete'),
    ),
  );
}
```

Функция называется `beep_action_info()`, поскольку, как и другие реализации хуков, для этого используется имя модуля (`beep`) с добавлением имени хука (`action_info`). Функция возвращает массив с элементами для каждого действия в модуле. Поскольку мы пишем лишь одно действие, в этом массиве будет только один элемент с ключом, содержащим имя функции, которая выполняет действие: `beep_beep_action()`. При чтении кода лучше сразу видеть, что функция представляет собой действие, поэтому к имени функции добавлен суффикс `_action` — в результате получилось `beep_beep_action()`.

Рассмотрим внимательнее ключи в нашем массиве.

- `type`. Указывает на вид создаваемого действия. Drupal использует эту информацию для разбивки действий по категориям в раскрывающемся списке выбора, который находится в пользовательском интерфейсе назначения триггера. К возмож-

ным типам относятся `system` (система), `node` (нода), `user` (пользователь), `comment` (комментарий) и `taxonomy` (таксономия). При определении типа создаваемого действия удобно задать вопрос: “С каким объектом работает это действие?” (Если ответ неочевиден или объекты могут быть различными, используйте тип `system`.)

- `label`. Это дружественное имя действия, отображаемое в раскрывающемся списке выбора, который находится в пользовательском интерфейсе назначения триггера.
- `configurable`. Указывает, принимает ли действие параметры.
- `triggers`. В массиве хуков каждый элемент должен перечислять операции, поддерживаемые действием. Drupal использует эту информацию для определения, стоит ли перечислять все возможные действия в пользовательском интерфейсе назначения триггера.

После описания нашего действия его можно реализовать:

```
/**
 * Действие Drupal — имитация гудка.
 */
function beep_beep_action() {
  beep_beep();
}
```

Совсем не трудно, так ведь? Прежде чем продолжить, удалите реализации `beep_user()` и `beep_nodeapi()`, поскольку мы будем использовать триггеры и действия, а не непосредственные реализации хуков.

Назначение действия

Теперь щелкните на ссылке **Structure** (Структура) в верхнем меню, а затем на странице **Structure** щелкните на ссылке **Triggers** (Триггеры). Если все сделано правильно, ваше действие должно быть доступно в пользовательском интерфейсе, как показано на рис. 3.2.

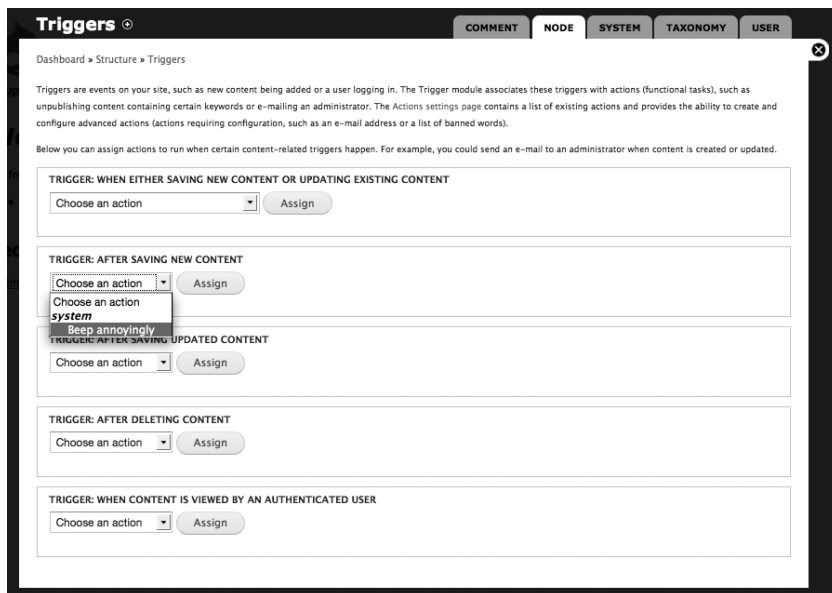


Рис. 3.2. Действие должно быть доступно для выбора в пользовательском интерфейсе триггеров

Назначьте действие триггеру, связанному с сохранением нового контента, для чего выберите пункт *Beep annoyingly* (Противно гудит) в раскрывающемся списке и щелкните на кнопке *Assign* (Назначить). После этого создайте новый элемент контента типа *Basic page* (Базовая страница) и сохраните его. После сохранения щелкните на ссылке *Reports* (Отчеты) в верхней части страницы и выберите отчет *Recent log entries* (Последние записи в журнале). Если вы правильно создали и настроили действие и триггер, должны появиться результаты, подобные приведенным на рис. 3.3.

	TYPE	DATE ▼	MESSAGE	USER	OPERATIONS
	content	05/12/2010 – 20:50	page: added Test Page.	admin	view
	beep	05/12/2010 – 20:50	Beep!	admin	

Рис. 3.3. Результатом действия со звуковым сигналом, запущенного при сохранении ноды, является запись в файле журнала

Изменение триггеров, поддерживаемых действием

При изменении значений, которые определяют операции, поддерживаемые данным действием, это изменение должно отразиться и в пользовательском интерфейсе. Например, если модифицировать код `beep_action_info()` так, как описано ниже, действие *Beep* будет доступно лишь триггеру *After deleting a node* (После удаления ноды):

```
/**
 * Реализация hook_action_info().
 */
function beep_action_info() {
  return array(
    'beep_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep annoyingly'), // Противно гудит
      'configurable' => FALSE,
      'triggers' => array('node_delete'),
    ),
  );
}
```

Действия, поддерживающие любые триггеры

Если вы не хотите ограничить свое действие каким-то конкретным триггером или набором триггеров, можете объявить, что данное действие поддерживает любые триггеры:

```
/**
 * Реализация hook_action_info().
 */
function beep_action_info() {
  return array(
    'beep_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep annoyingly'), // Противно гудит
      'configurable' => FALSE,
      'triggers' => array('any'),
    ),
  );
}
```

Расширенные действия

По сути, действия бывают двух видов: действия, принимающие параметры, и действия, которые параметров не принимают. Действие Веер, с которым мы работаем, не принимает параметров. При выполнении это действие выдает один звуковой сигнал, и все. Но зачастую действия должны быть несколько более специализированными. Например, действие Send e-mail (Отправить почтовое сообщение) должно знать адресата, тему и текст сообщения. Такие действия требуют некоторой настройки с помощью формы конфигурирования и называются *расширенными* (advanced) или *конфигурируемыми* (configurable) действиями.

Простые действия не принимают параметров, не требуют наличия формы конфигурирования и автоматически доступны системе. Чтобы сообщить Drupal, что создаваемое действие является расширенным, в реализации функции `hook_action_info()` понадобится установить ключ `configurable` в `TRUE`, построить форму для конфигурирования действия и создать необязательный обработчик проверки достоверности и обязательный обработчик отправки данных из формы конфигурирования. Различия между простым и расширенным действием представлены в табл. 3.3.

Таблица 3.3. Сводка различий простых и расширенных действий

	Простое действие	Расширенное действие
Параметры	Нет*	Требуются
Форма конфигурирования	Нет	Требуется
Доступность	Автоматически	Необходимо создание экземпляра действия с помощью страницы администрирования действий
Значение ключа <code>configure</code> в <code>hook_action_info()</code>	<code>FALSE</code>	<code>TRUE</code>

* При необходимости доступны параметры `$object` и `$context`

Давайте создадим расширенное действие, которое будет выдавать звуковой сигнал несколько раз. Конкретное количество гудков можно будет указывать с помощью формы конфигурирования. Сначала понадобится указать Drupal, что это действие является конфигурируемым. Добавим в реализацию хука `action_info` в модуле `beep.module` элемент для нового действия:

```
/**
 * Реализация hook_action_info().
 */
function beep_action_info() {
  return array(
    'beep_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep annoyingly'), // Противно гудит
      'configurable' => FALSE,
      'triggers' => array('node_view', 'node_insert', 'node_update', 'node_delete'),
    ),
    'beep_multiple_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep multiple times'), // Гудит несколько раз
      'configurable' => TRUE,
      'triggers' => array('node_view', 'node_insert', 'node_update', 'node_delete'),
    ),
  );
}
```

Быстро проверим, все ли правильно сделано — для этого нужно выбрать пункт меню **Administer**⇒**Site configuration**⇒**Actions** (Администрирование⇒Настройка сайта⇒Действия). Наше действие должно появиться в раскрывающемся списке расширенных действий, как показано на рис. 3.4.

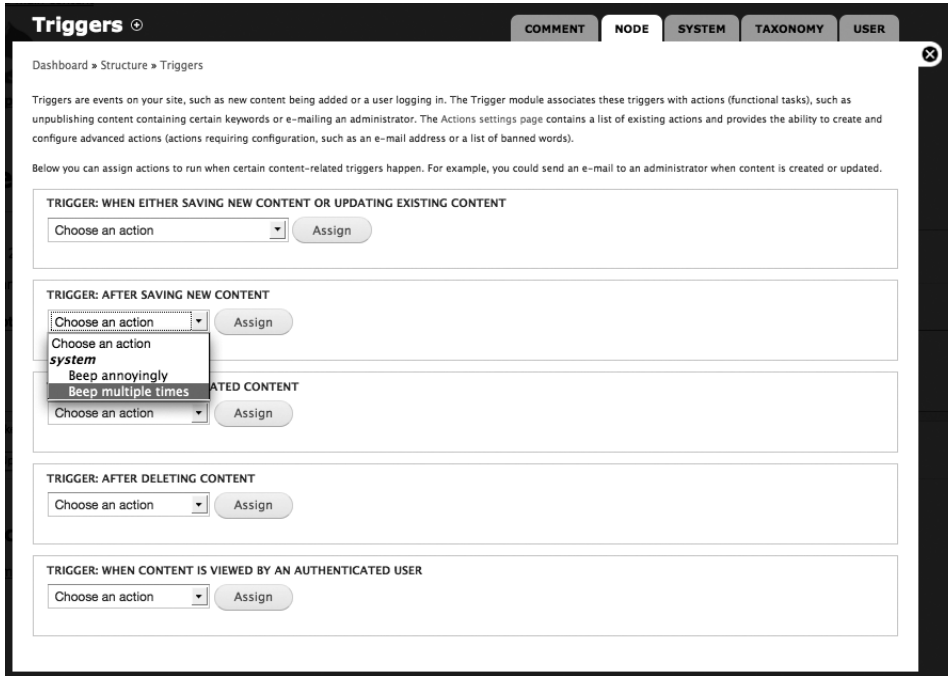


Рис. 3.4. Новое действие в списке расширенных действий

Теперь необходимо предоставить форму, в которой администратор сможет указать нужное количество звуковых сигналов. Для этого нужно определить одну или несколько форм с помощью API форм системы Drupal. Кроме того, понадобится написать функции для проверки форм и отправляемых данных. Имена этих функций основаны на идентификаторе действия, определенном в функции `hook_action_info()`. Для нашего действия идентификатор имеет вид `beep_multiple_beep_action`, и по соглашению к имени функции определения формы должно быть добавлено `_form` — получится `beep_multiple_beep_action_form`. В Drupal принято, что имя функции проверки образуется из идентификатора действия с добавлением суффикса `_validate` (`beep_multiple_beep_action_validate`), а имя функции отправки — с добавлением суффикса `_submit` (`beep_multiple_beep_action_submit`).

```
/**
 * Форма для конфигурируемого действия Drupal — выдачи нескольких звуковых сигналов.
 */
```

```
function beep_multiple_beep_action_form($context) {
  $form['beeps'] = array(
    '#type' => 'textfield',
    '#title' => t('Number of beeps'), // Количество гудков
    '#description' => t('Enter the number of times to beep when this action executes'),
    // Введите количество гудков при выполнении этого действия
    '#default_value' => isset($context['beeps']) ? $context['beeps'] : '1',
```

```

    '#required' => TRUE,
  );
  return $form;
}

function beep_multiple_beep_action_validate($form, $form_state) {
  $beeps = $form_state['values']['beeps'];
  if (!is_int($beeps)) {
    form_set_error('beeps', t('Please enter a whole number between 0 and 10.'));
    // Введите целое число от 0 до 10.
  }
  else if ((int) $beeps > 10 ) {
    form_set_error('beeps',
      t('That would be too annoying. Please choose fewer than 10 beeps.'));
    // Это будет слишком занудно. Укажите менее 10 сигналов.
  }
  else if ((int) $beeps < 0) {
    form_set_error('beeps',
      t('That would likely create a black hole! Beeps must be a positive integer.'));
    // Решили создать черную дыру?
    // Количество должно быть положительным целым числом.
  }
}

function beep_multiple_beep_action_submit($form, $form_state) {
  return array(
    'beeps' => (int)$form_state['values']['beeps']
  );
}

```

Первая функция создает описание формы для Drupal. В ней определено единственное текстовое поле, в котором администратор может вводить количество гудков. Чтобы открыть форму расширенного действия, щелкните на ссылке Configuration (Настройка) в верхней части страницы, а на открывшейся странице Configuration щелкните на ссылке Actions (Действия). Внизу страницы Actions откройте список выбора Create an Advanced action (Создать расширенное действие) и щелкните на пункте Beep multiple times (Гудит несколько раз). После выбора этого пункта Drupal отобразит форму настройки расширенного действия, показанную на рис. 3.5.

Configure an advanced action ✕

Dashboard » Configuration » System » Actions

An advanced action offers additional configuration options which may be filled out below. Changing the *Description* field is recommended, in order to better identify the precise action taking place. This description will be displayed in modules such as the Trigger module when assigning actions to system events, so it is best if it is as descriptive as possible (for example, "Send e-mail to Moderation Team" rather than simply "Send e-mail").

Label

Beep multiple times

A unique label for this advanced action. This label will be displayed in the interface of modules that integrate with actions, such as Trigger module.

Number of beeps *

1

Enter the number of times to beep when this action executes

Save

Рис. 3.5. Форма настройки действия Beep multiple times

В форму настройки действия среда Drupal добавила поле Description (Описание). Текст в этом поле можно изменить и использовать вместо стандартного описания, определенного в хуке `action_info`. Такие описания могут пригодиться в случаях, когда создается одно расширенное действие с двумя звуковыми сигналами и описанием “Два гудка” и другое действие с пятью звуковыми сигналами и описанием “Пять гудков”. Это позволяет различить два расширенных действия при их назначении триггеру. То есть расширенные действия можно снабжать описаниями, понятными администратору.

Совет. Эти два действия — “Два гудка” и “Пять гудков” — можно назвать экземплярами действия “Несколько гудков”.

Функция проверки достоверности ничем не отличается от других функций проверки форм в Drupal (подробно о проверке форм рассказывается в главе 10). В нашем случае нужно просто проверить, что пользователь действительно ввел число, и это число не слишком велико. Значение, возвращаемое функцией отправки данных для форм конфигурирования действий, имеет специальный вид. Оно должно быть массивом с ключами, содержащими имена необходимых полей. Значения в этом массиве доступны при выполнении действия. Описание обрабатывается автоматически, поэтому нужно только вернуть значение из введенного нами поля — т.е. количество звуковых сигналов.

Ну, наконец, пришло время написать само расширенное действие:

```
/**
 * Конфигулируемое действие. Воспроизводит указанное количество гудков.
 */
function beep_multiple_beep_action($object, $context) {
  for ($i = 1; $i < $context['beeps']; $i++) {
    beep_beep();
  }
}
```

Это действие принимает два параметра — `$object` и `$context`. В простом действии, которое было написано нами ранее, параметров не было.

На заметку! Простые действия могут принимать такие же параметры, как и конфигулируемые действия.

Поскольку PHP игнорирует параметры, которые передаются функции, но отсутствуют в сигнатуре функции, можно просто заменить сигнатуру нашего простого действия `beep_beep_action()` на `beep_beep_action($object, $context)`, и тогда можно будет кое-что узнать о текущем контексте. Все действия вызываются с параметрами `$object` и `$context`.

Использование контекста в действиях

Итак, мы знаем, что сигнатура функций для действий имеет вид `example_action($object, $context)`. Рассмотрим каждый из этих параметров.

- `$object`. Многие действия работают с одним из встроенных объектов Drupal: с нодами, пользователями, таксонометрическими терминами и т.д. При выполнении действия модулем `trigger.module` нужный объект передается действию в параметре `$object`. Например, если действие предназначено для выполнения при создании новой ноды, параметр `$object` будет содержать объект ноды.
- `$context`. Действие может быть выполнено во многих различных контекстах. Действия объявляют поддерживаемые ими триггеры с помощью определения ключа `hooks` в функции `hook_action_info()`. Но действиям, поддерживающим несколько триггеров, нужен способ определения контекста, в котором они вызваны, чтобы по-разному выполняться в зависимости от этого контекста.

Подготовка контекста модулем триггера

Предположим, что имеется веб-сайт, на котором возможны взаимоисключающие варианты. Вот один из коммерческих примеров: пользователи проходят платную регистрацию и могут оставлять на веб-сайте только один комментарий. После помещения одного комментария они блокируются и должны снова заплатить для разблокировки. Не задумываясь над экономическими перспективами такого сайта, рассмотрим, как можно реализовать необходимые триггеры и действия.

Нам понадобится действие, блокирующее текущего пользователя. Просмотрев модуль `user.module`, можно увидеть, в Drupal такое действие уже есть:

```
/**
 * Реализация hook_action_info().
 */
function user_action_info() {
  return array(
    'user_block_user_action' => array(
      'label' => t('Block current user'), // Блокировка текущего пользователя
      'type' => 'user',
      'configurable' => FALSE,
      'triggers' => array(),
    ),
  );
}
```

Однако это действие не видно на странице назначения триггеров, т.к. оно не объявляет ни одного поддерживаемого хука: `triggers` представляет собой просто пустой массив. Хорошо бы найти способ изменить это. И такой способ есть.

Изменение существующих действий с помощью `drupal_alter()`

При выполнении хука `action_info` Drupal позволяет каждому модулю не только объявить предоставляемые им действия, но и изменить эту информацию — в том числе и информацию, предоставляемую другими модулями. Вот как можно сделать действие `Block current user` (Блокировка текущего пользователя) доступным для триггера вставки комментария:

```
/**
 * Реализация hook_drupal_alter(). Вызывается Drupal после вызова
 * hook_action_info(), чтобы модули могли изменить массив action_info.
 *
 * @param array $info
 *   Результат вызова hook_action_info() для всех модулей.
 */
function beep_action_info_alter(&$info) {
  // Сделать действие Block current user доступным
  // для триггера вставки комментария.

  if (!in_array("comment_insert", $info['user_block_user_action']['triggers'])) {
    $info['user_block_user_action']['triggers'][] = 'comment_insert';
  }
}
```

После этого действие `Block current user` станет доступным для назначения, как показано на рис. 3.6.

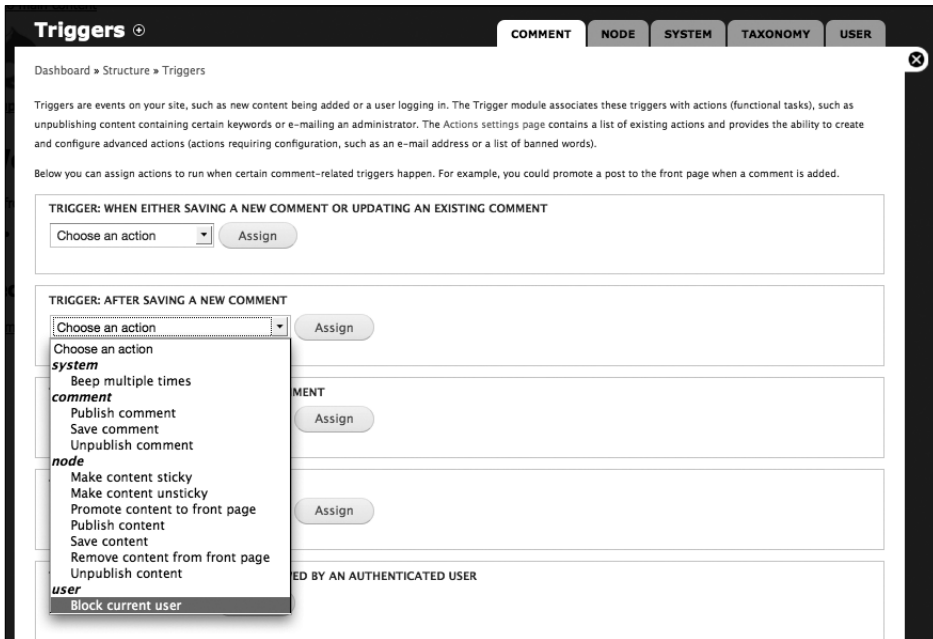


Рис. 3.6. Назначение действия Block current user триггеру вставки комментария

Формирование контекста

Когда назначено действие, текущий пользователь будет блокироваться после размещения нового комментария. Рассмотрим, что при этом происходит. Мы уже знаем, что Drupal уведомляет модули о возникновении определенных событий за счет запуска хука. В нашем случае это хук комментария. Конкретная выполняемая операция — `insert` (вставка), т.к. нужно реагировать на добавление нового комментария. Модуль триггера реализует хук комментария. Внутри такого хука выполняется запрос к базе данных, существуют ли действия, назначенные данному триггеру. База данных выдает информацию о назначенном нами действии `Block current user` (Блокировка текущего пользователя). Теперь модуль триггера готов к выполнению действия, которое имеет стандартную сигнатуру функции `example_action($object, $context)`.

Но тут возникает одна проблема. Действие, которое должно быть выполнено, имеет тип `user` (пользователь), а не `comment` (комментарий). Оно ожидает, что ему будет передан объект пользователя! В контексте хука комментария нужно вызвать действие пользователя, но хуку передается информация о комментарии, а не о пользователе. И как быть? Что произойдет, если модуль триггера определит, что действие нужно выполнить для пользователя, и загрузит объект `$user`, необходимый нашему действию? Вот код из модуля `modules/trigger/trigger.module`, который демонстрирует, как это происходит:

```
/**
 * Загрузка связанных объектов для триггеров комментария.
 *
 * При вызове действия в контексте, который не соответствует его типу, должен
 * быть извлечен объект, ожидаемый действием. Например, если хук комментария
 * вызывает действие, обрабатывающее ноды, объект ноды недоступен, т.к. он не
 * передается хуку комментария. Поэтому здесь загружается необходимый
 * действию объект.
```



```

*
* @param $type
*   Тип действия, которое должно быть выполнено.
* @param $comment
*   Комментарий, передаваемый через хук комментария.
*
* @return
*   Объект, нужный для действия, которое должно быть выполнено.
*/
function _trigger_normalize_comment_context($type, $comment) {
  switch ($type) {
    // Действие для работы с нодами вызвано в контексте комментария.
    case 'node':
      return node_load(is_array($comment) ? $comment['nid'] : $comment->nid);

    // Действие для работы с пользователями вызвано в контексте комментария.
    case 'user':
      return user_load(is_array($comment) ? $comment['uid'] : $comment->uid);
  }
}

```

При выполнении этого кода для нашего действия пользователя срабатывает второй вариант: загружается объект пользователя и затем выполняется действие пользователя. Информация, известная хуку комментария (например, тема комментария), передается действию в параметре `$context`. Обратите внимание, что действие ищет идентификатор пользователя сначала в объекте, потом в контексте, и затем переходит к глобальной информации в `$user`:

```

/**
 * Блокировка текущего пользователя.
 *
 * @ingroup actions
 */
function user_block_user_action(&$entity, $context = array()) {
  if (isset($entity->uid)) {
    $uid = $entity->uid;
  }
  elseif (isset($context['uid'])) {
    $uid = $context['uid'];
  }
  else {
    global $user;
    $uid = $user->uid;
  }
  db_update('users')
    ->fields(array('status' => 0))
    ->condition('uid', $uid)
    ->execute();
  drupal_session_destroy_uid($uid);
  watchdog('action', 'Blocked user %name.', array('%name' => $user->name));
}

```

Действия должны вести себя интеллектуально, поскольку они не очень-то знают, что происходит при их вызове. Поэтому лучшие действия выглядят просто и даже являются атомарными. Модуль триггера всегда передает текущий хук и действие в контекст. Эти значения хранятся в элементах `$context['hook']` и `$context['op']`. Такой подход предлагает стандартный способ предоставления информации действию.

Хранение действий

Действия — это функции, которые выполняются в указанный момент. Для простых действий не нужны конфигурируемые параметры. Например, созданное ранее действие Веер просто выдает звуковой сигнал. Ему не нужна никакая дополнительная информация (хотя, разумеется, параметры `$object` и `$context` доступны). Сравните это действие с также созданным нами расширенным действием Веер multiple times (Несколько гудков): ему необходимо знать, сколько нужно звуковых сигналов. Другим расширенным действиям, вроде Send e-mail (Послать почтовое сообщение), может потребоваться еще более информации: адресат, тема сообщения и т.д. Эти параметры должны храниться в базе данных.

Таблица actions

Когда администратор создает экземпляр расширенного действия, информация, введенная в форме конфигурирования, сериализуется и сохраняется в поле `parameters` таблицы `actions`. Запись для простого действия Веер может выглядеть следующим образом:

```
aid: 2
type: 'system'
callback: 'beep_beep_action'
parameters: (сериализованный массив, содержащий параметр beeps и его
              значение, т.е. количество гудков)
description: Beep three times
```

Перед выполнением расширенного действия содержимое поля `parameters` десериализуется и заносится в параметр `$context`, который передается действию. Так что количество гудков в экземпляре действия Веер multiple times доступно для функции `beep_multiple_beep.action()` как `$context['beeps']`.

Идентификаторы действий

В предыдущем разделе обратите внимание на различие в идентификаторах действий для двух записей таблиц. Идентификатор простого действия — это имя выполняющей его функции. Понятно, что имя функции невозможно использовать в качестве идентификаторов расширенных действий, т.к. могут храниться несколько экземпляров одного и того же действия. Поэтому в таких случаях применяются числовые идентификаторы действия (храняемые в таблице базы данных `actions_aid`).

Механизм выполнения действий определяет, нужно ли выполнять процесс извлечения хранимых параметров, проверяя, является ли идентификатор действия числовым. Если он не числовой, действие выполняется без привлечения базы данных. Эта проверка производится очень быстро; Drupal использует такой же подход и в `index.php`, чтобы отличить контент от константных значений меню.

Непосредственный вызов действия с помощью actions_do()

Модуль триггера — это лишь один из способов вызова действий. Может понадобиться написать отдельный модуль, который сам вызывает действия и подготавливает для них параметры. В таком случае для вызова действий рекомендуется функция `actions_do()`. Ее сигнатура имеет следующий вид:

```
actions_do($action_ids, $object = NULL, $context = NULL, $a1 = NULL, $a2 = NULL)
```

Рассмотрим каждый из ее параметров.

- `$action_ids`. Действие или действия, которые нужно выполнить; задаются одним идентификатором действия или массивом таких идентификаторов.
- `$object`. Объект, над которым будет выполняться действие: нода, пользователь, комментарий или просто никакой.
- `$context`. Ассоциативный массив, содержащий информацию, которая может потребоваться для выполнения действия, в том числе настроенные параметры для расширенных действий.
- `$a1` и `$a2`. Необязательные дополнительные параметры; если они переданы в `actions_do()`, то они передаются и действию.

Ниже показано, как можно вызвать простое действие Веер с помощью функции `actions_do()`:

```
$object = NULL; // параметр $object обязателен, хотя в данном случае не используется
actions_do('beep_beep_action', $object);
```

А вот так вызывается расширенное действие Веер *multiple times*:

```
$object = NULL;
actions_do(2, $object);
```

Но можно выполнить и следующий вызов с обходом извлечения хранимых параметров:

```
$object = NULL;
$context['beeps'] = 5;
actions_do('beep_multiple_beep_action', $object, $context);
```

На заметку! Искушенные разработчики на PHP могут удивиться: зачем вообще нужны действия? Разве нельзя просто вызвать функцию или реализовать хук? Зачем морочиться с сохранением параметров в контексте, а потом снова выбирать их вместо использования обычных параметров PHP? Ответ состоит в том, что написание действий с весьма обобщенной сигнатурой функции позволяет поручить повторное использование кода администратору сайта. Этот администратор может и не знать PHP, но ему не обязательно звонить разработчику PHP, скажем, для настройки отправления почтового сообщения при добавлении нового кода. Администратор сайта просто свяжет действие “Отправить почтовое сообщение” с триггером, который срабатывает при сохранении новой ноды, и ему не потребуется никого звать на помощь.

Определение собственных триггеров с помощью `hook_trigger_info()`

Каким образом Drupal узнает, какие триггеры доступны для вывода в пользовательском интерфейсе? Обычно Drupal позволяет модулям определять хуки с объявлениями, какие триггеры реализованы в этих модулях. Вот, к примеру, простая реализация функции `hook_trigger_info()` из самого модуля `Triggers`, который определяет все стандартные триггеры, доступные после установки ядра Drupal 7:

```
/**
 * Реализация hook_trigger_info().
 *
 * Определение всех триггеров.
 */
function trigger_trigger_info() {
  return array(
```

```

'node' => array(
  'node_presave' => array(
    'label' => t('When either saving new content or updating existing content'),
    // При сохранении нового контента или обновлении существующего контента
  ),
  'node_insert' => array(
    'label' => t('After saving new content'),
    // После сохранения нового контента
  ),
  'node_update' => array(
    'label' => t('After saving updated content'),
    // После сохранения измененного контента
  ),
  'node_delete' => array(
    'label' => t('After deleting content'),
    // После удаления контента
  ),
  'node_view' => array(
    'label' => t('When content is viewed by an authenticated user'),
    // При просмотре контента аутентифицированным пользователем
  ),
),
'comment' => array(
  'comment_presave' => array(
    'label' => t('When either saving a new comment or updating existing comment'),
    // При сохранении нового комментария или изменении существующего комментария
  ),
  'comment_insert' => array(
    'label' => t('After saving a new comment'),
    // После сохранения нового комментария
  ),
  'comment_update' => array(
    'label' => t('After saving an updated comment'),
    // После сохранения измененного комментария
  ),
  'comment_delete' => array(
    'label' => t('After deleting a comment'),
    // После удаления комментария
  ),
  'comment_view' => array(
    'label' => t('When a comment is being viewed by an authenticated user'),
    // При просмотре комментария аутентифицированным пользователем
  ),
),
'taxonomy' => array(
  'taxonomy_term_insert' => array(
    'label' => t('After saving a new term to the database'),
    // После сохранения нового термина в базе данных
  ),
  'taxonomy_term_update' => array(
    'label' => t('After saving an updated term to the database'),
    // После сохранения измененного термина в базе данных
  ),
  'taxonomy_term_delete' => array(
    'label' => t('After deleting a term'), // После удаления термина
  ),
),

```

```

'system' => array(
  'cron' => array(
    'label' => t('When cron runs'),          // При запуске cron
  ),
),
'user' => array(
  'user_presave' => array(
    'label' => t('When either creating a new user account or updating an existing'),
    // При создании новой пользовательской учетной записи или изменении существующей
  ),
  'user_insert' => array(
    'label' => t('After creating a new user account'),
    // После создания новой пользовательской учетной записи
  ),
  'user_update' => array(
    'label' => t('After updating a user account'),
    // После изменения пользовательской учетной записи
  ),
  'user_delete' => array(
    'label' => t('After a user has been deleted'),
    // После удаления пользователя
  ),
  'user_login' => array(
    'label' => t('After a user has logged in'),
    // После входа пользователя
  ),
  'user_logout' => array(
    'label' => t('After a user has logged out'),
    // После выхода пользователя
  ),
  'user_view' => array(
    'label' => t('When a user's profile is being viewed'),
    // При просмотре пользовательского профиля
  ),
),
);
}

```

Как видно по структуре функции, каждая категория триггера (т.е. `node`, `comment`, `system` и `user`) возвращает массив опций, которые отображаются на странице конфигурирования триггеров. В каждом таком массиве категории определяется триггер и метка, которая отображается на странице настройки триггеров, наподобие приведенной ниже, где `node_insert` — имя триггера, а значение, связанное с элементом метки — это то, что отображается на странице *Trigger configuration* (Конфигурация триггера).

```

'node_insert' => array(
  'label' => t('After saving new content'),
)

```

Если добавить в модуль аннотации из главы 2 хуки, то эти хуки могут выглядеть следующим образом:

```

/**
 * Реализация hook_trigger_info().
 */
function annotate_trigger_info() {
  return array(

```

```

'annotate' => array(
  'annotate_insert' => array(
    'label' => t('After saving new annotations'),
    // После сохранения новых аннотаций
  ),
  'annotate_update' => array(
    'label' => t('After saving updated annotations'),
    // После сохранения измененных аннотаций
  ),
  'annotate_delete' => array(
    'label' => t('After deleting annotations'),
    // После удаления аннотаций
  ),
  'annotate_view' => array(
    'label' => t('When annotation is viewed by an authenticated user'),
    // При просмотре аннотаций аутентифицированным пользователем
  ),
),
);
}

```

После очистки кэша Drupal возьмет новую реализацию `hook_trigger_info()` и изменит страницу триггеров, т.е. добавит отдельную вкладку для хука `Annotate`, как показано на рис. 3.7. Конечно, сам модуль все так же отвечает за инициацию хуков с помощью `module_invoke()` или `module_invoke_all()` и за запуск выполнения действий. В нашем примере модулю нужно выполнить вызов `module_invoke_all('annotate_insert', 'annotate_update', 'annotate_delete', 'annotate_view')`. Тогда потребуется реализовать `hook_annotate_insert`, `hook_annotate_update`, `hook_annotate_delete` или `hook_annotate_view` и запускать действия с помощью `actions_do()`.

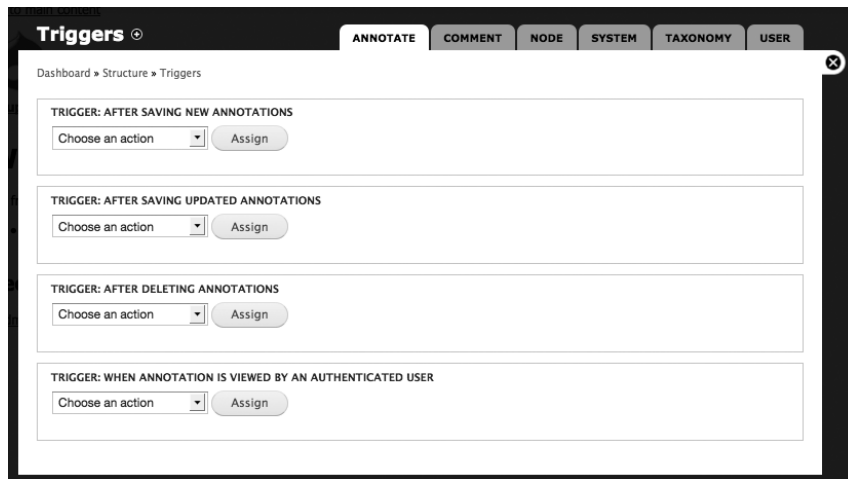


Рис. 3.7. Для только что определенного триггера создается новая вкладка в пользовательском интерфейсе триггера

Добавление триггеров в существующие хуки

Иногда бывает нужно добавить триггеры в уже существующие хуки, если в коде добавляется новая операция. Например, может понадобиться добавить хук по имени `hook_node_archive`. Предположим, что уже написан модуль архивации, который выби-

рает старые ноды и перемещает их в хранилище данных. Для этого можно определить полностью новый хук, и это вполне нормально. Но поскольку операция выполняется с нодой, можно запускать хук `hook_node_archive`, чтобы все триггеры для контента находились на одной вкладке в интерфейсе триггеров. При условии, что модуль называется `archive` (архивация), следующий код добавляет дополнительный триггер:

```
/**
 * Реализация hook_trigger_info().
 */
function archive_trigger_info() {
  return array(
    'node' => array(
      'archive_nodes' => array(
        'label' => t('Archive old nodes'), // Архивация старых нод
      )
    )
  );
}
```

Теперь новый триггер доступен в списке триггеров на странице администрирования триггеров (рис. 3.8).

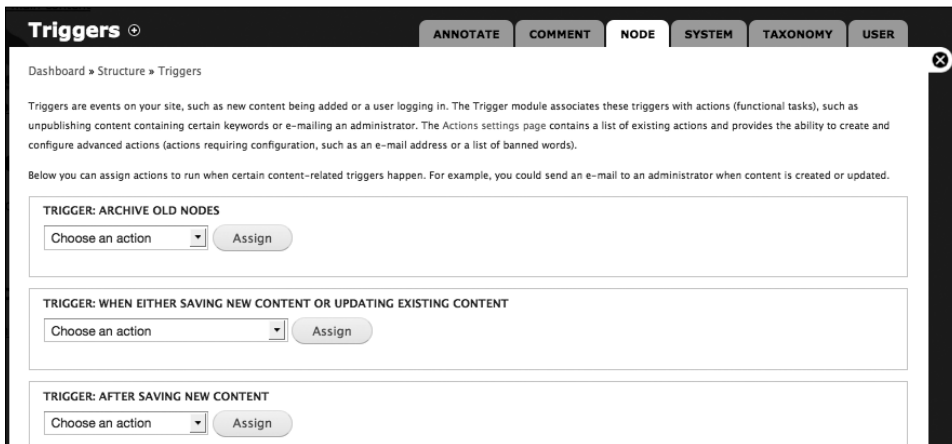


Рис. 3.8. Дополнительный триггер `Archive old nodes` в пользовательском интерфейсе

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Знать, как назначать действия триггерам.
- Писать простые действия.
- Писать расширенные действия и связанные с ними формы конфигурирования.
- Создавать экземпляры расширенных действий с помощью страницы администрирования действий и изменять их имена.
- Понимать, что такое контекст.
- Понимать, как можно использовать контекст для изменения поведения действия.
- Знать, как хранятся, извлекаются и выполняются действия.
- Определять собственные хуки и отображать их в виде триггеров.

ГЛАВА 4

Система меню

Система меню Drupal является сложной, зато мощной. Можно считать, что она отвечает за три основные функции: отображение обратных вызовов, управление доступом и настройка меню. Код самой системы меню находится в файле `includes/menu.inc`, а дополнительный код с такими средствами, как настройка меню, расположен в `modules/menu`.

В настоящей главе мы рассмотрим, что такое отображение обратных вызовов и как оно работает, узнаем, как защитить элементы меню с помощью управления доступом, научимся использовать в меню обобщенные символы и рассмотрим различные встроенные типы элементов меню. В конце главы будет показано, как переопределять, добавлять и удалять существующие элементы меню, чтобы сделать Drupal как можно более ненавязчивым.

Отображение обратных вызовов

Когда веб-браузер отправляет запрос к Drupal, он предоставляет некоторый URL-адрес. Исходя из этой информации, Drupal может определить, какой код необходимо выполнить и как обработать запрос. Весь этот процесс обычно называется *маршрутизацией* (routing) или *диспетчеризацией* (dispatching). Drupal усекает базовую часть URL и использует последнюю его часть, которая называется *путем* (path). Например, для URL-адреса `http://example.com/?q=node/3` путь Drupal имеет вид `node/3`. При использовании средства чистых URL адресом в браузере будет `http://example.com/node/3`, но веб-сервер молча переписывает URL в виде `http://example.com/?q=node/3`, поэтому Drupal всегда работает с одинаковым путем. В предыдущем примере путь Drupal имеет вид `node/3` независимо от того, включено ли средство чистых URL. Этот механизм описан в разделе “Роль веб-сервера” главы 1.

Отображение URL-адресов на функции

Общий подход таков: Drupal запрашивает у всех активированных модулей предоставления массива *элементов меню*. Каждый элемент меню содержит массив, индексированный путем и содержащий информацию о пути. Один из фрагментов информации, который должен предоставить модуль — это *обратный вызов страницы* (page callback). В данном контексте это просто имя PHP-функции, которая будет выполнена при запросе браузером определенного пути. При поступлении запроса Drupal выполняет следующие шаги.

1. Установка пути Drupal. Если путь является псевдонимом реального пути, Drupal определяет этот реальный путь и использует его. Предположим, что администратор присвоил пути `http://example.com/?q=node/3` псевдоним `http://example.com/?q=about` (например, с помощью модуля пути), тогда Drupal использует в качестве пути `node/3`.

2. Drupal отслеживает соответствие путей и обратных вызовов в таблице базы данных `menu_router` и отслеживает элементы меню, которые являются ссылками, в таблице `menu_links`. Выполняется проверка необходимости перестройки таблиц `menu_router` и `menu_links`, хотя это требуется редко — после инсталляции или обновления Drupal.
 3. Определяется, какой элемент таблицы `menu_router` соответствует пути Drupal, и создается элемент маршрутизатора, описывающий обратный вызов, который необходимо выполнить.
 4. Загружаются все необходимые объекты, которые нужно передать обратному вызову.
 5. Выполняется проверка, разрешен ли пользователю доступ к обратному вызову. Если нет, возвращается сообщение “Access denied” (Доступ запрещен).
 6. Выполняется локализация заголовка и описания элемента меню для текущего языка.
 7. Загружаются все необходимые включаемые файлы.
 8. Выполняется обратный вызов и возвращается результат, который `index.php` передает через функцию `theme_page()`, что и дает окончательную веб-страницу.
- Визуальное представление этого процесса показано на рис. 4.1 и 4.2.

Создание элемента меню

Для создания элемента меню предназначена функция `hook_menu()`. Она принимает массив элементов, которые необходимо добавить в меню, и каждый такой элемент сам представляет собой массив пар “ключ-значение”, определяющих атрибуты элемента меню. Ключи массива элементов меню приведены в табл. 4.1.

Перехват управления процессом выполняется с помощью хука меню в модуле. Это позволяет определить элементы меню, которые будут включены в таблицу маршрутизатора. Давайте построим пример модуля по имени `menufun.module`, чтобы поэкспериментировать с системой меню. Мы выполним отображение пути Drupal `menufun` на PHP-функцию `menufun_hello()`, которую также предстоит написать. Вначале необходим файл `menufun.info` по адресу `sites/all/modules/custom/menufun/menufun.info`:

```
name = Menu Fun
description = Learning about the menu system.
package = Pro Drupal Development
core = 7.x
files[] = menufun.module
```

После этого понадобится создать файл `sites/all/modules/custom/menufun/menufun.module`, содержащий реализацию `hook_menu()` и функции, которая должна выполняться.

```
<?php
/**
 * @file
 * Модуль для работы с системой меню Drupal.
 */
/**
 * Реализация hook_menu().
 */
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Greeting',
    'page callback' => 'menufun_hello',
    'access callback' => TRUE,
    'type' => MENU_CALLBACK,
  );
  return $items;
}
```

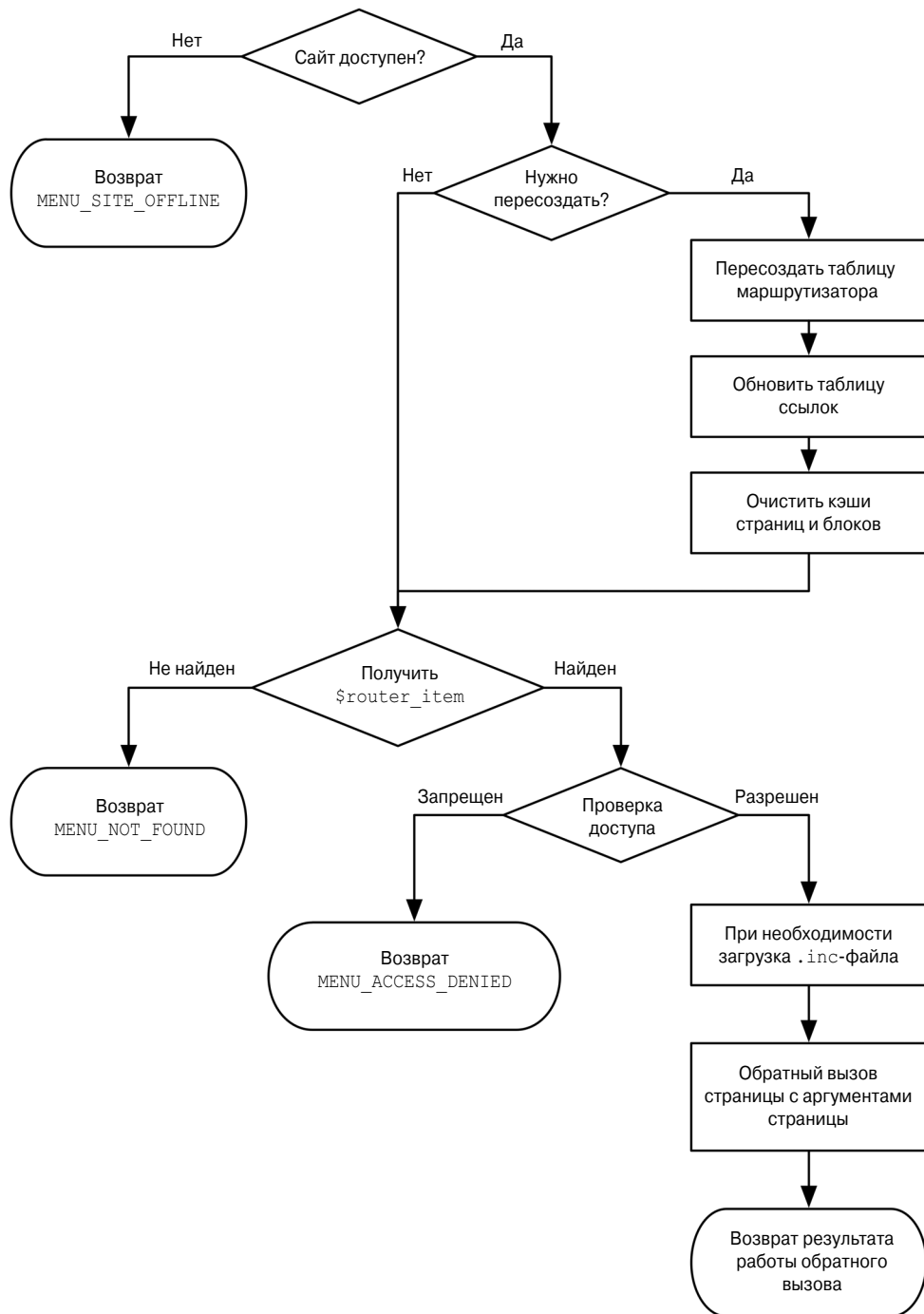


Рис. 4.1. Укрупненная блок-схема процесса диспетчеризации меню

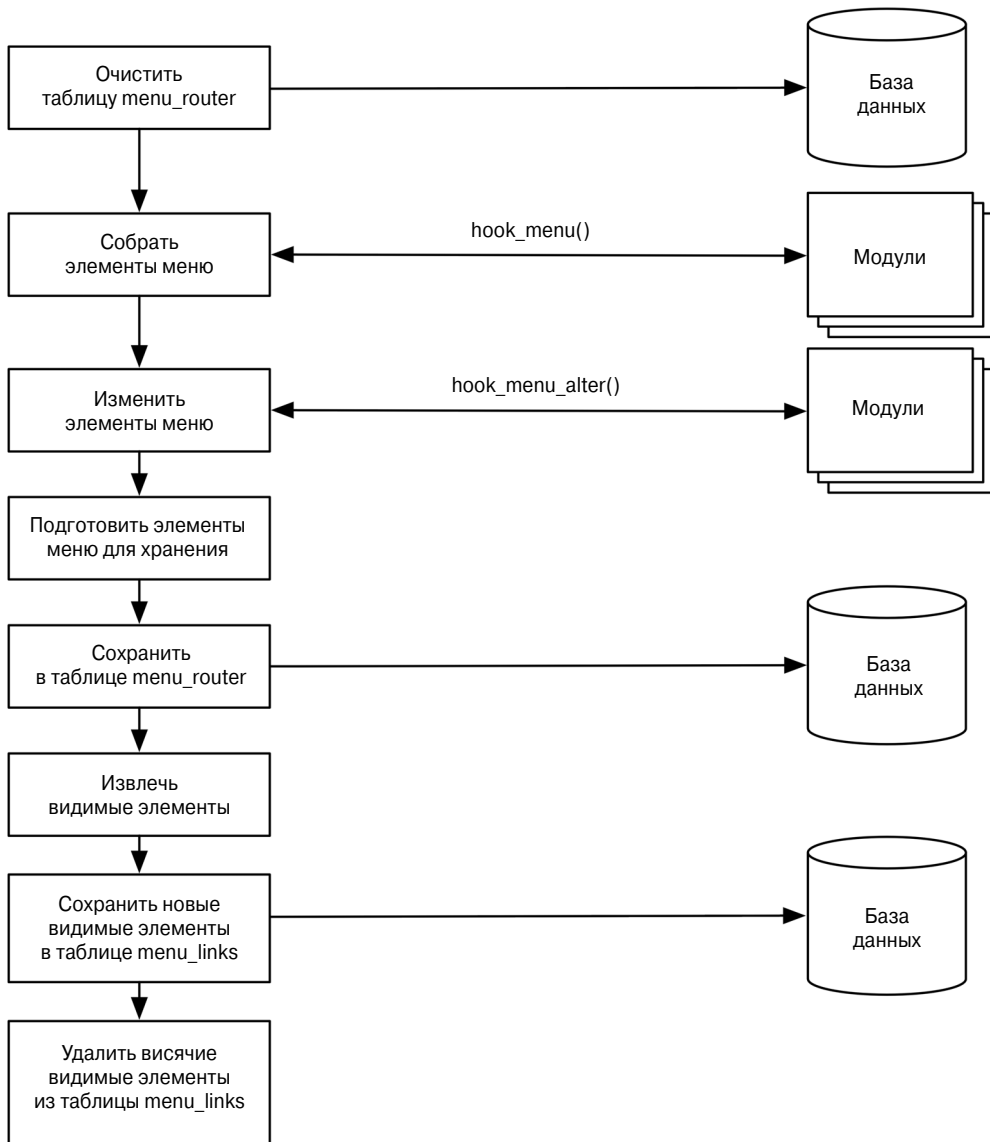


Рис. 4.2. Укрупненная блок-схема процесса построения маршрутизации и ссылок

Таблица 4.1. Атрибуты “ключ-значение” для функции `hook_menu()`

Ключ	Значение
<code>title</code> (заголовок)	Обязательное поле, которое представляет неперебиваемый заголовок элемента меню
<code>title callback</code> (обратный вызов заголовка)	Функция, используемая для генерации заголовка. По умолчанию это функция <code>t()</code> , поэтому заголовок (<code>title</code>) не нужно упаковывать в функцию <code>t()</code> . Если заголовок переводить не нужно, укажите значение <code>FALSE</code>
<code>title arguments</code> (аргументы заголовка)	Аргументы, которые нужно передать функции <code>t()</code> или собственному обратному вызову
<code>description</code> (описание)	Непереведенное описание элемента меню
<code>page callback</code> (обратный вызов страницы)	Функция, которую нужно вызвать, чтобы вывести веб-страницу, когда пользователь посещает путь
<code>page arguments</code> (аргументы страницы)	Массив аргументов, которые нужно передать функции обратного вызова страницы; целые значения передают соответствующий компонент URL-адреса
<code>access callback</code> (обратный вызов доступа)	Функция, возвращающая логическое значение, которое определяет, имеет ли пользователь право доступа к этому элементу меню; по умолчанию это <code>user_access()</code> , если значение не унаследовано от родительского элемента меню
<code>access arguments</code> (аргументы доступа)	Массив аргументов, которые нужно передать функции обратного вызова доступа; целые значения передают соответствующий компонент URL-адреса
<code>file</code> (файл)	Файл, который будет включен перед обратными вызовами; это позволяет хранить функции обратных вызовов в отдельных файлах. Файл должен быть указан относительно каталога реализующего модуля, если не задано что-то другое в параметре <code>file path</code>
<code>file path</code> (путь файла)	Путь к папке, содержащей файл, который указан в параметре <code>file</code> . По умолчанию это каталог с модулем, реализующим хук
<code>weight</code> (вес)	Целое число, определяющее относительное положение элемента в меню; элементы с большим весом находятся ниже. По умолчанию равно 0. Если вы не знаете, какое число присвоить, не задавайте это значение: обычно вполне годится стандартный алфавитный порядок
<code>menu_name</code> (имя меню)	Необязательный параметр, с помощью которого можно указать собственное меню, если не нужно, чтобы элемент находился в меню навигации
<code>type</code> (тип)	Битовое сочетание флагов, которые описывают свойства элемента меню; используются следующие значения: <code>MENU_NORMAL_ITEM</code> — обычные элементы меню помещаются в дерево меню и могут быть перемещены или скрыты администратором; <code>MENU_CALLBACK</code> — обратные вызовы просто регистрируют путь, чтобы при обращении к URL вызывалась нужная функция; <code>MENU_SUGGESTED_ITEM</code> — модули могут “предлагать” элементы меню, доступные для активации администратором; <code>MENU_LOCAL_TASK</code> — по умолчанию локальные задачи отображаются в виде вкладок; <code>MENU_DEFAULT_LOCAL_TASK</code> — каждый набор локальных задач должен содержать одну “стандартную” задачу, которая при щелчке выполняет переход к родительскому пути.

Приведенный код создает наше меню (`$items['menufun']`) за счет создания массива с тремя парами “ключ-значение”:

```
title — обязательное значение, которое определяет непереуведенный заголовок
элемента меню;
page callback — функция, которая будет вызываться при посещении пользова-
телем пути меню;
access callback — обычно содержит функцию, которая возвращает логическое
значение.
/**
 * Обратный вызов страницы.
 */
function menufun_hello() {
  return t('Hello!');
}
```

Активация данного модуля приводит к вставке элемента меню в таблицу маршрутизатора, после чего Drupal может найти и выполнить функцию при переходе по адресу `http://example.com/?q=menufun`, как показано на рис. 4.3.

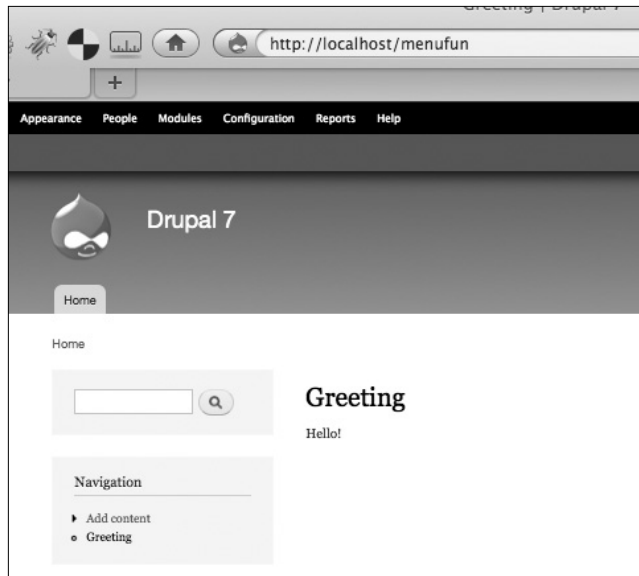


Рис. 4.3. Элементу меню разрешено найти и выполнить в Drupal функцию `menufun_hello()`

Обратите внимание, что мы определяем путь и отображаем его на функцию. Путь — это путь Drupal. Он определяется в виде ключа в массиве `$items`. Мы используем путь, совпадающий с именем модуля — это позволяет не затрагивать пространство имен URL. Но в принципе можно определить любой путь.

Аргументы обратного вызова страницы

Иногда функции обратного вызова страницы, которая отображена на путь, требуется предоставить больше информации. Прежде всего, все дополнительные части пути передаются автоматически. Изменим нашу функцию следующим образом:

74 Глава 4. Система меню

```
function menufun_hello($first_name = '', $last_name = '') {  
    return t('Hello @first_name @last_name',  
        array('@first_name' => $first_name, '@last_name' => $last_name));  
}
```

Если теперь перейти по адресу `http://example.com/?q=menufun/John/Doe`, то мы увидим на экране то, что показано на рис. 4.4.

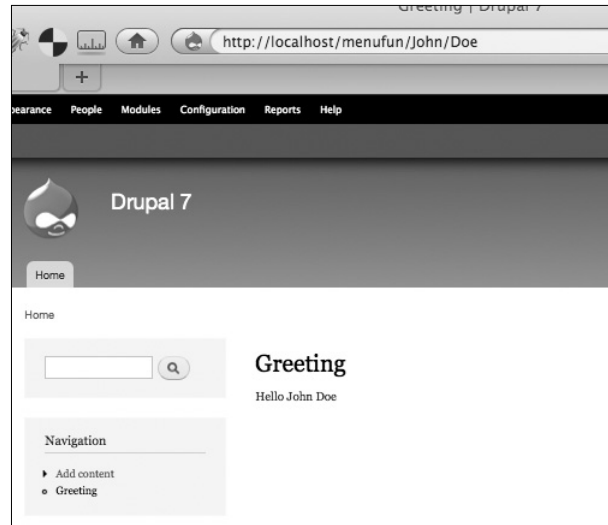


Рис. 4.4. Части пути переданы функции обратного вызова

Обратите внимание, что каждый дополнительный компонент URL передан в виде параметра функции обратного вызова.

В хуке меню можно также определить аргументы обратного вызова страницы, добавив в массив `$items` дополнительный ключ аргументов страницы (*page arguments*). Определение аргумента страницы удобно тем, что позволяет получить больший контроль над параметрами, передаваемыми функции обратного вызова.

В качестве примера модифицируем модуль `menufun`, добавив в него аргументы страницы для нашего элемента меню:

```
function menufun_menu() {  
    $items['menufun'] = array(  
        'title' => 'Greeting',  
        'page callback' => 'menufun_hello',  
        'page arguments' => array('Jane', 'Doe'),  
        'access callback' => TRUE,  
        'type' => MENU_CALLBACK,  
    );  
    return $items;  
}
```

После того как Drupal выполнит все инструкции, которые явно заданы в качестве аргументов страницы, в функцию обратного вызова страницы передаются все оставшиеся аргументы страницы в виде дополнительных параметров — с помощью средства перегрузки параметров для функций в PHP. Аргументы из URL все так же видны; для обращения к ним нужно изменить сигнатуру функции обратного вызова — добавить параметры из URL. Поэтому для нашего переделанного элемента меню будет актуальна

следующая сигнатура функции, где `$first_name` содержит значение Jane (из первого элемента массива `page arguments`), а `$last_name` — значение Doe (из второго элемента массива `page arguments`):

```
function menufun_hello($first_name = '', $last_name = '') {...}
```

Проверим поведение элемента меню: поместим Jane Doe в аргументы меню, а John Doe — в URL-адрес, и посмотрим, что получится. Теперь переход по адресу `http://example.com/?q=menufun/John/Doe` даст результаты, показанные на рис. 4.5 (если у вас это не получилось, значит, вы забыли перестроить меню).

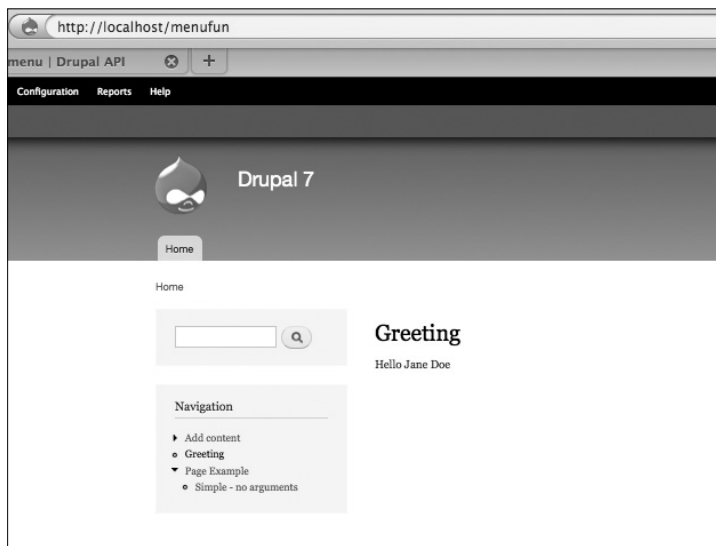


Рис. 4.5. Передача и вывод аргументов в функции обратного вызова

Чтобы использовать переданные значения в URL, можно изменить функцию обратного вызова следующим образом:

```
function menufun_hello($first_name = '', $last_name = '') {
  return t('Hello @first_name @last_name from @from_first_name @from_last_name',
    Привет от array('@first_name' => $first_name, '@last_name' => $last_name));
}
```

Обновите свою версию, очистите кэш и посмотрите, что получится при переходе по адресу `http://example.com/?q=menufun`.

Обратные вызовы страниц в других файлах

Если не указано иное, Drupal предполагает, что обратный вызов страницы находится в файле `.module`. В Drupal 7 многие модули разбиты на несколько файлов, которые загружаются в зависимости от определенных условий — это позволяет при каждом запросе страницы загружать минимальный объем кода. Ключ `file` (например, `'file' => 'menufun_greetings.inc'`) элемента меню используется для указания имени файла, который содержит функцию обратного вызова.

В качестве примера мы включим в функцию `menufun.module hook_menu()` имя файла, в котором находится новая функция обратного вызова. Следующий код добавляет пару `'file' => 'menufun_greetings'` в массив элемента меню.

Кроме того, здесь обратный вызов страницы изменен на `menufun_greeting` — просто для демонстрации того, что обратный вызов не использует функцию, уже существующую в файле `menufun.module`.

```
/**
 * Реализация hook_menu().
 */
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Menu Fun',
    'page callback' => 'menufun_greeting',
    'file' => 'menufun_greeting.inc',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => TRUE,
    'type' => MENU_CALLBACK,
  );
  return $items;
}
```

Теперь создадим в каталоге `menufun` новый файл по имени `menufun_greeting.inc` со следующим кодом внутри:

```
<?php

function menufun_greeting($first_name = '', $last_name = '', $from_first_name='',
$from_last_name='') {
  return t('Hello @first_name @last_name from @from_first_name @from_last_name',
    array('@first_name' => $first_name, '@last_name' => $last_name,
      '@from_first_name' => $from_first_name,
      '@from_last_name' => $from_last_name));
}
```

Сохраните оба файла, очистите кэш и проверьте, что получилось теперь. Вы должны получить точно такие же результаты, но на этот раз функция обратного вызова находится во внешнем файле `.module`.

Добавление ссылки в блок навигации

В примере `menufun` мы объявили, что наш элемент меню имеет тип `MENU_CALLBACK`. Изменение этого типа на `MENU_NORMAL_ITEM` укажет, что мы не просто хотим выполнить отображение пути на функцию обратного вызова, а желаем также, чтобы среда Drupal включила его в меню.

```
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Menu Fun',
    'page callback' => 'menufun_greeting',
    'file' => 'menufun_greeting.inc',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => TRUE,
    'type' => MENU_NORMAL_ITEM,
  );
  return $items;
}
```

Теперь элемент меню появится в блоке навигации (рис. 4.6).

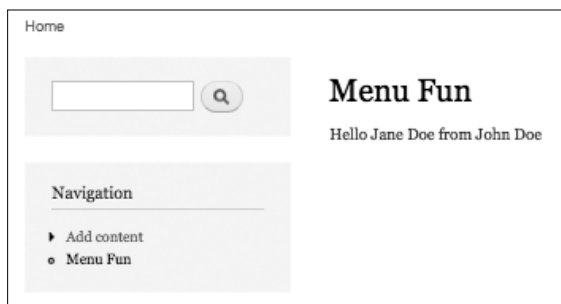


Рис. 4.6. Элемент меню в блоке навигации

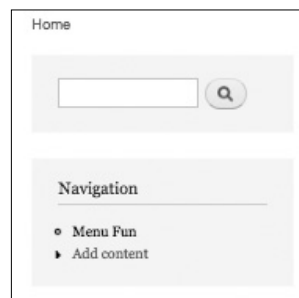


Рис. 4.7. Элементы меню с большим весом расположены ниже в блоке навигации

Если не нравится его расположение, можно переместить его вверх или вниз, уменьшая или увеличивая его вес. Вес — это один из ключей в определении элемента меню (weight):

```
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Greeting',
    'page callback' => 'menufun_hello',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => TRUE,
    'weight' => -1,
  );
  return $items;
}
```

Эффект такого уменьшения веса продемонстрирован на рис. 4.7. Элементы меню могут быть также перемещены и без изменения кода — с помощью средств администрирования меню, которые доступны через пункт меню Structure ⇒ Menus (Структура ⇒ Меню) (для этого необходимо активировать модуль меню).

Вложенные меню

До сих пор мы определили только один статический элемент меню. Добавим еще один элемент и еще один обратный вызов для него:

```
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Menu Fun',
    'page callback' => 'menufun_greeting',
    'file' => 'menufun_greeting.inc',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => TRUE,
    'type' => MENU_NORMAL_ITEM,
    'weight' => '-1',
  );
  $items['menufun/farewell'] = array(
    'title' => 'Farewell',
    'page callback' => 'menufun_farewell',
    'file' => 'menufun_greeting.inc',
    'access callback' => TRUE,
    'type' => MENU_NORMAL_ITEM,
  );
  return $items;
}
```

Теперь добавим в файл `menufun_greeting.inc` функцию обратного вызова `menufun_farewell`:

```
function menufun_farewell() {
    return t('Goodbye');
}
```

После изменения модуля не забудьте очистить кэш.

Среда Drupal заметит, что путь второго элемента меню (`menufun/farewell`) является дочерним по отношению к пути первого элемента меню (`menufun`). Поэтому при визуализации меню (в формате HTML) Drupal сдвинет второй элемент, как показано на рис. 4.8. Кроме того, будет правильно настроена навигационная цепочка в верхней части страницы, указывая на вложенность элемента меню. Разумеется, тема может визуализировать меню и цепочки так, как пожелает разработчик.

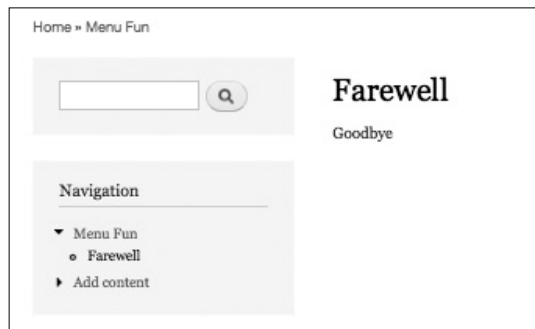


Рис. 4.8. Вложенное меню

Управление доступом

Во всех приведенных выше примерах мы просто заносили в ключ `access callback` элемента меню значение `TRUE`, а это означает, что к данному элементу может обратиться любой пользователь. Обычно управление доступом к меню осуществляется за счет определения прав доступа в модуле с помощью функции `hook_permission()` и последующей проверки этих прав. Имя необходимой для этого функции определено в ключе `access callback` и обычно выглядит как `user_access`. Определим право доступа с именем `receive greeting`; если у пользователя нет роли с этим правом, то при попытке перехода по адресу `http://example.com/?q=menufun` он получит сообщение `Access denied` (Доступ запрещен).

```
/**
 * Реализация hook_permission()
 */
function menufun_permission() {
    return array(
        'receive greeting' => array(
            'title' => t('Receive a greeting'),
            'description' => t('Allow users receive a greeting message'),
        ),
    );
}
/**
 * Реализация hook_menu().
 */
```

```
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Menu Fun',
    'page callback' => 'menufun_greeting',
    'file' => 'menufun_greeting.inc',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => 'user_access',
    'access arguments' => array('receive greeting'),
    'type' => MENU_NORMAL_ITEM,
    'weight' => '-1',
  );

  $items['menufun/farewell'] = array(
    'title' => 'Farewell',
    'page callback' => 'menufun_farewell',
    'file' => 'menufun_greeting.inc',
    'access callback' => 'user_access',
    'access arguments' => array('receive greeting'),
    'type' => MENU_NORMAL_ITEM,
  );

  return $items;
}
```

В этом коде право доступа определяется с помощью вызова `user_access('receive greeting')`. Здесь система меню работает в качестве стража, определяя на основе пользовательской роли, к каким путям разрешено обращаться, а к каким нет.

Совет. Функция `user_access()` является стандартным обратным вызовом доступа. Если не определить обратный вызов доступа, то система меню передаст аргументы доступа функции `user_access()`.

Дочерние элементы меню не наследуют обратные вызовы доступа и аргументы доступа от своих родителей.

Ключ `access arguments` должен быть определен для каждого элемента меню. Ключ `access callback` нужно определять, только если он отличается от `user_access`. Исключением является любой элемент меню типа `MENU_DEFAULT_LOCAL_TASK`, который наследует родительские ключи `access callback` и `access arguments`, хотя для ясности лучше явно определять и эти ключи даже для стандартных локальных задач.

Локализация и настройка заголовка

Существуют два вида заголовков: статические и динамические. Статические заголовки создаются с помощью функции обратного вызова заголовка. Drupal автоматически переводит значения статических заголовков, поэтому помещать их в вызов `t()` не требуется. Однако при использовании динамических заголовков с помощью функции обратного вызова заголовка за перевод в обратном вызове отвечаете вы.

```
'title' => t('Greeting') // Не используйте t() в заголовках
                        // и описаниях элементов меню!
```

На заметку! Описания всегда статичны, определяются значением ключа описания (`description`) и автоматически переводятся системой Drupal.

Определение обратного вызова заголовка

Заголовки можно создавать динамически во время выполнения с помощью обратного вызова заголовка. В приведенном ниже примере демонстрируется использование функции обратного вызова заголовка, которая заносит в заголовок текущую дату и время. Из-за применения обратного вызова заголовка функция отвечает за выполнение перевода перед возвратом значения. Для реализации перевода возвращаемое значение упаковывается в вызов `t()`.

```
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Greeting',
    'title callback' => 'menufun_title',
    'description' => 'A salutation.',
    'page callback' => 'menufun_hello',
    'access callback' => TRUE,
  );
  return $items;
}
/**
 * Обратный вызов страницы.
 */
function menufun_hello() {
  return t('Hello!');
}
/**
 * Обратный вызов заголовка.
 */
function menufun_title() {
  $now = format_date(time());
  return t('It is now @time', array('@time' => $now));
}
```

Как показано на рис. 4.9, заголовок элемента меню во время выполнения можно установить с помощью пользовательского обратного вызова заголовка.

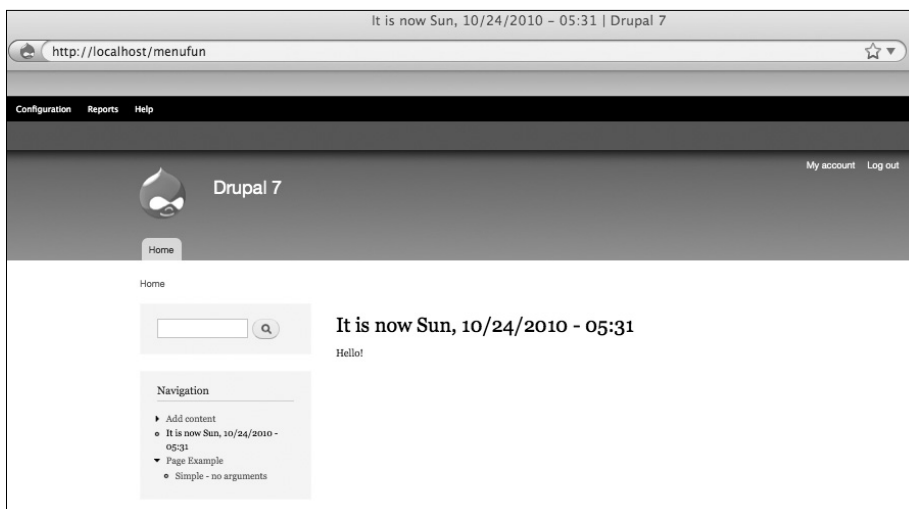


Рис. 4.9. Обратный вызов заголовка для установки заголовка элемента меню

А что если нужно отделить заголовок элемента меню от заголовка страницы? Это нетрудно сделать с помощью вызова `drupal_set_title()`:

```
function menufun_title() {
  drupal_set_title(t('The page title'));
  $now = format_date(time());
  return t('It is now @time', array('@time' => $now));
}
```

В этом случае получается один заголовок для страницы и другой — для элемента меню, как показано на рис. 4.10.

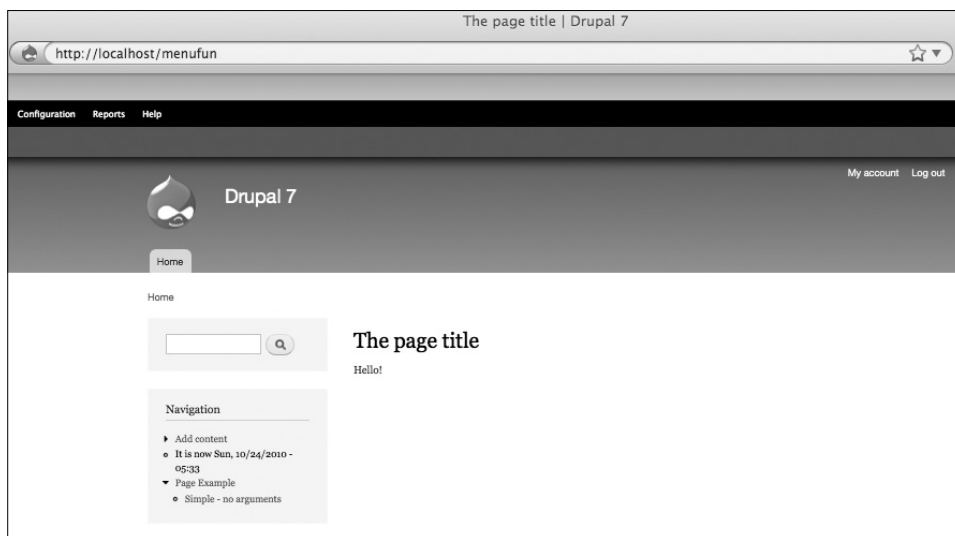


Рис. 4.10. Отдельные заголовки для элемента меню и страницы

Обобщенные символы в элементах меню

До сих пор мы использовали в элементах меню обычные имена путей Drupal, такие как `menufun` и `menufun/farewell`. Однако в Drupal часто применяются пути вида `user/4/track` или `node/15/edit`, где части пути формируются динамически. Рассмотрим этот процесс подробнее.

Основные обобщенные символы

Символ `%` играет в элементах меню Drupal роль обобщенного символа и означает, что точное значение определяется во время выполнения на основе символа в той позиции URL, где находится обобщенный символ. Вот элемент меню, в котором применяется обобщенный символ:

```
function menufun_menu() {
  $items['menufun/%'] = array(
    'title' => 'Hi',
    'page callback' => 'menufun_hello',
    'access callback' => TRUE,
  );
  return $items;
}
```

Этот элемент меню срабатывает для путей Drupal `menufun/hi`, `menufun/foo/bar`, `menufun/123` и `menufun/file.html`. Однако он *не* будет работать для пути `menufun` — для такого пути необходимо написать отдельный элемент меню, потому что он содержит только одну часть, а выражение `menufun/%` соответствует лишь строкам из двух частей. Символ `%` часто применяется для обозначения числа (как в `user/%/edit` для `user/2375/edit`), однако он соответствует любому тексту в данной позиции.

На заметку! Элемент меню с обобщенным символом в пути не отображается в навигационных меню, даже если тип элемента меню задан как `MENU_NORMAL_ITEM`. И понятно, почему: поскольку путь содержит обобщенный символ, Drupal не знает, как сконструировать URL для такой ссылки. Способ указания Drupal, какой URL следует использовать, описан ниже в данной главе, в разделе “Сборка путей из обобщенных символов с помощью функций `to_arg()`”.

Обобщенные символы и параметры обратного вызова страницы

Обобщенный символ в конце пути меню не мешает передаче дополнительных путей URL в обратный вызов пути, поскольку обобщенный символ соответствует тексту только до ближайшей наклонной черты. Вернемся к примеру пути `menufun/%`: в адресе `http://example.com/?q=menufun/foo/Fred` строка `foo` соответствует обобщенному символу, а последняя часть пути (`Fred`) передается в качестве параметра в обратный вызов страницы.

Использование значения обобщенного символа

Чтобы использовать соответствующую часть пути, нужно указать номер этой части в ключе `page arguments`:

```
function menufun_menu() {
  $items['menufun/%/bar/baz'] = array(
    'title' => 'Hi',
    'page callback' => 'menufun_hello',
    'page arguments' => array(1), // Фрагмент, соответствующий обобщенному символу
    'access callback' => TRUE,
  );
  return $items;
}

/**
 * Обратный вызов страницы.
 */
function menufun_hello($name = NULL) {
  return t('Hello. $name is @name', array('@name' => $name));
}
```

Параметры, переданные функции обратного вызова страницы `menufun_hello()`, будут такими, как показано на рис. 4.11.

Первый параметр `$name` передается с помощью обратного вызова страницы. Элемент `array(1)` для обратного вызова страницы означает: “передать часть 1 пути, какой бы она ни была”. Нумерация начинается с 0, так что частью 0 будет `menufun`, частью 1 — то, что соответствует обобщенному символу, частью 2 — `bar` и т.д. Второй параметр `$b` передается в соответствии с правилом передачи в Drupal части пути за пределами пути Drupal в качестве параметра (см. раздел “Аргументы обратного вызова страницы” ранее в главе).

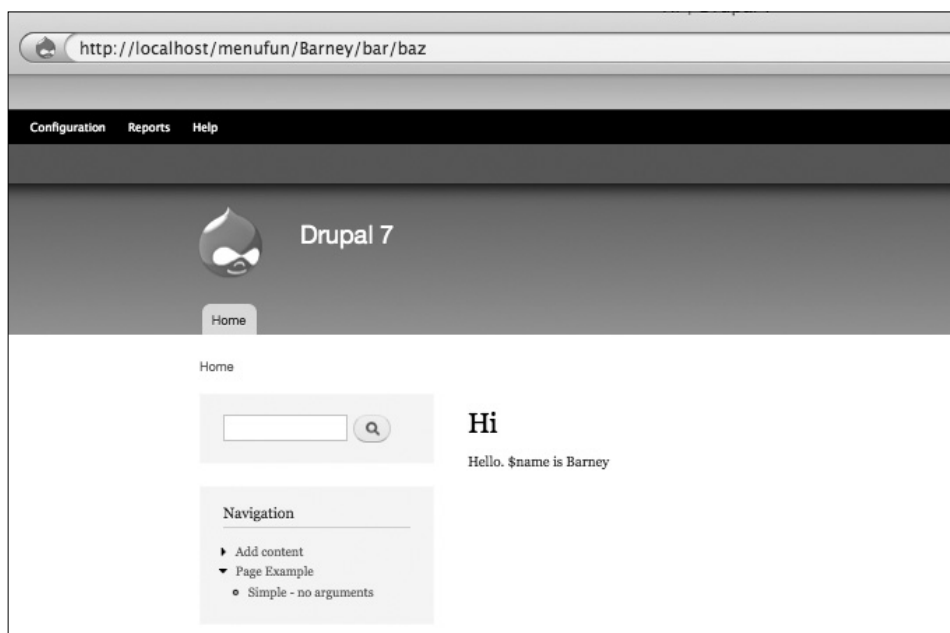


Рис. 4.11. Первый параметр взят из фрагмента, соответствующего обобщенному символу

Обобщенные символы и замена параметров

На практике части пути Drupal обычно используются для просмотра или изменения какого-либо объекта — вроде ноды или пользователя. Например, путь `node/%/edit` применяется для редактирования ноды, а путь `user/%` — для просмотра информации о пользователе по его идентификатору. Рассмотрим элемент меню для последнего случая, который можно найти в реализации `hook_menu()` в модуле `modules/user/user.module`. URL, соответствующий данному пути, будет иметь вид, подобный `http://example.com/?q=user/2375`. Такой URL предназначен для того, чтобы, щелкнув на нем, увидеть страницу **My account** (Моя учетная запись) на сайте Drupal.

```
$items['user/%user_uid_optional'] = array(
  'title' => 'My account',
  'title callback' => 'user_page_title',
  'title arguments' => array(1),
  'page callback' => 'user_view_page',
  'page arguments' => array(1),
  'access callback' => 'user_view_access',
  'access arguments' => array(1),
  'weight' => -10,
  'menu_name' => 'user-menu',
);
```

Когда среда Drupal создает меню на основе пути `user/%user_uid_optional`, она заменяет фрагмент `%user_uid_optional` согласно описанному ниже процессу.

1. Во втором сегменте находится строка, соответствующая `%` с последующими символами до ближайшей наклонной черты. В данном случае это `user_uid_optional`.
2. К этой строке добавляется `_load` для получения имени функции. В данном случае именем функции будет `user_uid_optional_load`.

3. Эта функция вызывается с передачей ей в качестве аргумента значения обобщенного символа из пути Drupal. Если URL выглядит как `http://example.com/?q=user/2375`, путем Drupal является `user/2375`, и обобщенный символ соответствует второму сегменту — `2375`. Поэтому производится вызов `user_uid_optional_load('2375')`.
4. Затем результат этого вызова используется *вместо* обобщенного символа. Поэтому при обратном вызове заголовка с аргументами заголовка `array(1)` вместо передачи части 1 пути Drupal (`2375`) передается результат вызова `user_uid_optional_load('2375')` — т.е. объект пользователя. Этот процесс можно рассматривать как замену части пути Drupal соответствующим объектом.
5. В обратных вызовах страницы и доступа также используется объект замены. Поэтому в предыдущем элементе меню будет выполнен вызов `user_view_access()` для доступа и `user_view()` для генерации контента страницы, и в обоих вызовах будет передан объект пользователя `2375`.

Совет. При замене объектов в пути Drupal вида `node/%node/edit` удобнее рассматривать часть `%node` как обобщенный символ с аннотацией непосредственно в строке. То есть `node/%node/edit` — это `node/%/edit` с неявным указанием выполнения `node_load()` при обнаружении соответствия обобщенному символу.

Передача дополнительных аргументов функции загрузки

Если функции загрузки необходимо передать дополнительные аргументы, их можно определить в ключе `load arguments`. Ниже приведен пример из модуля `node` — элемент меню для просмотра ревизии ноды. И идентификатор ноды, и идентификатор ревизии необходимо передать функции загрузки — в данном случае `node_load()`.

```
$items['node/%node/revisions/%/view'] = array(
  'title' => 'Revisions',
  'load arguments' => array(3),
  'page callback' => 'node_show',
  'page arguments' => array(1, TRUE),
  'access callback' => '_node_revision_access',
  'access arguments' => array(1),
);
```

В элементе меню для ключа `load arguments` указано значение `array(3)`. Это означает, что в дополнение к значению обобщенного символа для идентификатора ноды, который передается функции загрузки автоматически, как было описано выше, передается еще один параметр, поскольку массив `array(3)` содержит один элемент — целое число 3. Как было показано в разделе “Использование значения обобщенного символа”, это означает, что используется часть пути в позиции 3. Позиция и аргументы пути для примера URL `http://example.com/?q=node/56/revisions/4/view` приведены в табл. 4.2.

Таблица 4.2. Позиция и аргументы для пути Drupal `node/%node/revisions/%/view` при просмотре страницы `http://example.com/?q=node/56/revisions/4/view`

Позиция	Аргумент	Значение из URL
0	node	node
1	%node	56
2	revisions	revisions
3	%	4
4	view	view

Таким образом, при определении ключа `load arguments` вместо вызова `node_load('56')` выполняется вызов `node_load('56', '4')`. При выполнении обратного вызова функция загрузки заменит значение '56' загруженным объектом ноды, поэтому будет выполнен обратный вызов страницы `node_show($node, NULL, TRUE)`.

Специальные предопределенные аргументы загрузки %map и %index

Существуют два специальных аргумента загрузки. Элемент `%map` передает текущий путь Drupal в виде массива. В предыдущем примере значением `%map`, которое передается в качестве аргумента загрузки, будет `array('node', '56', 'revisions', '4', 'view')`. Значения отображения можно изменять с помощью функции загрузки, если в ней параметр объявлен как ссылочный. Поэтому в предыдущем примере значение элемента будет равно 1, т.к. обобщенный символ находится в позиции 1 (см. табл. 4.2).

Сборка путей из обобщенных символов с помощью функций to_arg()

Ранее было сказано, что Drupal не может создать верную ссылку из пути Drupal, если он содержит обобщенный символ вроде `user/%` — ведь Drupal не знает, чем заменить символ %. Однако это не совсем так. Можно определить вспомогательную функцию, которая выполняет замену обобщенного символа, и Drupal будет использовать эту замену при построении ссылки. В элементе меню `My account` (Моя учетная запись) путь для ссылки `My account` формируется с помощью следующих шагов.

1. Первоначальный путь Drupal — `user/%user_uid_optional`.
2. При построении ссылки Drupal ищет функцию с именем `user_uid_optional_to_arg()`. Если такая функция не определена, Drupal не может выяснить, как построить путь, и не выводит ссылку.
3. Если функция определена, Drupal использует результат ее вызова для замены обобщенного символа в ссылке. Функция `user_uid_optional_to_arg()` возвращает идентификатор текущего пользователя, поэтому если это пользователь 4, то Drupal связывает ссылку `My account` с адресом `http://example.com/?q=user/4`.

Использование функции `to_arg()` не ограничивается лишь выполнением заданного пути: она вызывается при построении ссылки на любой странице, а не только на конкретной странице, которая соответствует пути Drupal для элемента меню. Ведь ссылка `My account` выводится на всех страницах, а не только при просмотре страницы `http://example.com/?q=user/3`.

Специальные варианты обобщенных символов и функции to_arg()

Функция `to_arg()`, которую ищет Drupal при построении ссылки для элемента меню, основывается на строке, следующей за обобщенным символом в пути Drupal. Это может быть любая строка, например:

```
/**
 * Реализация hook_menu().
 */
function menufun_menu() {
  $items['menufun/%a_zoo_animal'] = array(
    'title' => 'Hi',
    'page callback' => 'menufun_hello',
    'page arguments' => array(1),
    'access callback' => TRUE,
```

```

    'type' => MENU_NORMAL_ITEM,
    'weight' => -10
  );
  return $items;
}
function menufun_hello($animal) {
  return t("Hello $animal");
}
function a_zoo_animal_to_arg($arg) {
  // $arg содержит '%', т.к. это обобщенный символ.
  // Заменяем его животным из зоопарка.
  return 'tiger';
}

```

Этот фрагмент помещает в блок навигации ссылку Hi с URL-адресом `http://example.com/?q=menufun/tiger`. Обычно обобщенный символ не заменяется статической строкой, как в этом простом примере. Взамен функция `to_arg()` используется для создания чего-то динамического, вроде идентификатора текущего пользователя или текущей ноды.

Изменение элементов меню из других модулей

При перестройке таблицы `menu_router` и изменении таблиц `menu_link` (например, во время активации нового модуля) модули имеют возможность изменить любой элемент меню, реализовав функцию `hook_menu_alter()`. Например, элемент меню Log off (Выход) выполняет выход текущего пользователя с помощью вызова `user_logout()`, который уничтожает сеанс этого пользователя, а затем перенаправляет его на домашнюю страницу сайта.

Функция `user_logout()` находится в файле `modules/user/user.pages.inc`, поэтому в элементе меню для пути Drupal определен ключ `file`. Так что обычно Drupal загружает файл `modules/user/user.pages.inc` и выполняет обратный вызов страницы `user_logout()` при щелчке пользователя на ссылке Log out (Выход) в блоке навигации. Изменим это поведение, чтобы перенаправлять пользователя после выхода на сайт `drupal.org`:

```

/**
 * Реализация hook_menu_alter().
 *
 * @param array $items
 * Элементы меню, снабженные ключами из пути.
 */
function menufun_menu_alter(&$items) {
  // Замена обратного вызова 'user_logout' обращением к нашему обратному вызову
  $items['logout']['page callback'] = 'menufun_user_logout';
  $items['logout']['access callback'] = 'user_is_logged_in';
  // Теперь Drupal не нужно загружать файл user.pages.inc,
  // т.к. будет вызываться функция menufun_user_logout(),
  // которая находится в нашем модуле - и уже в области видимости.
  unset($items['logout']['file']);
}
/**
 * Обратный вызов меню; выполняет выход текущего пользователя
 * и его перенаправление на drupal.org.
 * Это измененная версия user_logout().
 */

```

```
function menufun_user_logout() {
  global $user;

  watchdog('menufun', 'Session closed for %name.', array('%name' => $user->name));

  // Уничтожение текущего сеанса:
  session_destroy();

  // Выполнение операции logout хука user, чтобы при
  // необходимости модули могли отреагировать на выход.
  module_invoke_all('user', 'logout', NULL, $user);

  // Загрузка анонимного пользователя, чтобы глобальный объект
  // $user был корректным для всех реализаций hook_exit().
  $user = drupal_anonymous_user();

  drupal_goto('http://drupal.org/');
}
```

До выполнения нашей реализации `hook_menu_alter()` элемент меню для пути выхода имел следующий вид:

```
array(
  'access callback' => 'user_is_logged_in',
  'file'            => 'user.pages.inc',
  'module'          => 'user',
  'page callback'   => 'user_logout',
  'title'           => 'Log out',
  'weight'          => 10,
)
```

А после его изменения обратный вызов страницы заменен `menufun_user_logout`:

```
array(
  'access callback' => 'user_is_logged_in',
  'module'          => 'user',
  'page callback'   => 'menufun_user_logout',
  'title'           => 'Log out',
  'weight'          => 10,
)
```

Изменение ссылок меню из других модулей

Когда Drupal сохраняет элемент меню в таблице `menu_link`, модулям предоставляется возможность изменить ссылку с помощью реализации `hook_menu_link_alter()`. Ниже показано, как заменить надпись элемента меню Log out на Sign off:

```
/**
 * Реализация hook_link_alter().
 *
 * @param $item
 * Ассоциативный массив с определением ссылки меню,
 * передаваемой в menu_link_save()
 */
function menufun_menu_link_alter(&$item) {
  if ($item['link_path'] == 'user/logout') {
    $item['link_title'] = 'Sign off';
  }
}
```

Этот хук следует использовать для заголовка или веса ссылки. При необходимости изменить другие свойства элемента меню (например, обратный вызов доступа), вместо него должен быть вызван `hook_menu_alter()`.

На заметку! Изменения, внесенные а элемент меню в `hook_menu_link_alter()`, невозможно переопределить из пользовательского интерфейса, который модуль `menu.module` формирует на странице Administer⇒Site building⇒Menus (Администрирование⇒Структура сайта⇒Меню).

Виды элементов меню

При добавлении элемента меню в хуке меню можно использовать ключ `type` (тип). Если тип не определен, то используется стандартный тип `MENU_NORMAL_ITEM`. В зависимости от назначенного типа Drupal по-разному воспринимает элемент меню. Каждый такой тип элемента меню состоит из ряда *флагов*, или атрибутов (см. `includes/menu.inc`). Флаги типов для элементов меню перечислены в табл. 4.3.

Таблица 4.3. Флаги типов для элементов меню

Двоичный	Шестнадцатеричный	Десятичный	Константа	Описание
000000000001	0x0001	1	<code>MENU_IS_ROOT</code>	Элемент меню является корнем дерева меню
000000000010	0x0002	2	<code>MENU_VISIBLE_IN_TREE</code>	Элемент меню видим в дереве меню
000000000100	0x0004	4	<code>MENU_VISIBLE_IN_BREADCRUMB</code>	Элемент меню видим в навигационной цепочке
000000001000	0x0008	8	<code>MENU_LINKS_TO_PARENT</code>	Элемент меню указывает на родительский элемент
000000100000	0x0020	32	<code>MENU_MODIFIED_BY_ADMIN</code>	Элемент меню может быть изменен администратором
000001000000	0x0040	64	<code>MENU_CREATED_BY_ADMIN</code>	Элемент меню создан администратором
000010000000	0x0080	128	<code>MENU_IS_LOCAL_TASK</code>	Элемент меню является локальной задачей
000100000000	0x0100	256	<code>MENU_IS_LOCAL_ACTION</code>	Элемент меню является локальным действием

Например, константа `MENU_NORMAL_ITEM` (`define('MENU_NORMAL_ITEM', MENU_VISIBLE_IN_TREE | MENU_VISIBLE_IN_BREADCRUMB)`) содержит флаги `MENU_VISIBLE_IN_TREE` и `MENU_VISIBLE_IN_BREADCRUMB`, как показано в табл. 4.4. Поэтому тип `MENU_NORMAL_ITEM` содержит флаги 000000000110. В табл. 4.5 приведены доступные типы для элементов меню и соответствующие им флаги.

Таблица 4.4. Флаги типа для элемента меню `MENU_NORMAL_ITEM`

Двоичный	Константа
000000000010	<code>MENU_VISIBLE_IN_TREE</code>
000000000100	<code>MENU_VISIBLE_IN_BREADCRUMB</code>
000000000110	<code>MENU_NORMAL_ITEM</code>

Таблица 4.5. Флаги, соответствующие типам элементов меню

Флаги меню	Константы типа меню					
	MENU_NORMAL_ITEM	MENU_CALLBACK	MENU_SUGGESTED_ITEM*	MENU_LOCAL_TASK	MENU_DEFAULT_LOCAL_TASK	
MENU_IS_ROOT						
MENU_VISIBLE_IN_TREE	✓					
MENU_VISIBLE_IN_BREADCRUMB	✓		✓	✓	✓	
MENU_LINKS_TO_PARENT					✓	
MENU_MODIFIED_BY_ADMIN						
MENU_CREATED_BY_ADMIN						
MENU_IS_LOCAL_TASK			✓	✓	✓	

* Эта константа сформирована с помощью дополнительного битового "ИЛИ" с константой 0x0010.

Так какую константу использовать при определении типа элемента меню? Посмотрите в табл. 4.5, решите, какие флаги нужны в вашем случае, и выберите константу, содержащую эти флаги. Подробное описание всех констант содержится в комментариях в файле `includes/menu.inc`. Чаще всего применяются `MENU_CALLBACK`, `MENU_LOCAL_TASK` и `MENU_DEFAULT_LOCAL_TASK`.

Часто встречающиеся задачи

В настоящем разделе рассмотрены некоторые типичные подходы к часто встречающимся задачам, которые приходится решать разработчикам при работе с меню.

Назначение обратных вызовов без добавления ссылки в меню

Часто бывает необходимо создать отображение URL-адреса на функцию без создания видимого элемента меню. Например, веб-форма может иметь функцию JavaScript, которой нужно получить из Drupal список состояний — поэтому требуется связать URL и PHP-функцию, но все это не должно отображаться ни в одном навигационном меню. Это делается назначением элементу меню типа `MENU_CALLBACK`, как в первом примере в данной главе.

Вывод элементов меню в виде вкладок

Обратный вызов, который отображается в виде вкладки, называется *локальной задачей* (local task) и имеет тип `MENU_LOCAL_TASK` или `MENU_DEFAULT_LOCAL_TASK`. Заголовок локальной задачи должен быть коротким глаголом, таким как “add” (добавить) или “list” (перечислить). Локальные задачи ведут себя как некоторая разновидность объектов, вроде ноды или пользователя. Их можно считать семантическими явлениями элемента меню, который обычно отображается в виде вкладки — примерно так же, как дескриптор `<strong>` является семантическим объявлением и обычно отображается как полужирный шрифт.

Для отображения вкладок у локальных задач *должен быть* родительский элемент. Обычно назначается обратный вызов корневому пути вроде `milkshake`, а затем локальные задачи назначаются путям, расширяющим этот путь — вроде `milkshake/prepare`, `milkshake/drink` и т.п. В Drupal имеется встроенная поддержка тем для двух уровней локальных задач в виде вкладок. (Базовая система поддерживает и дополнительные уровни, но тогда тема должна обеспечить поддержку их отображения.)

Порядок отображения вкладок определяется алфавитным упорядочением заголовков элементов меню. Если он вас не устраивает, элементам меню можно назначить весовые ключи, и тогда элементы будут упорядочены по весам.

Приведенный ниже пример содержит код для вывода двух главных вкладок и двух вложенных вкладок для стандартной локальной задачи. Создайте файл `sites/all/modules/custom/milkshake/milkshake.info` со следующим содержимым:

```
name = Milkshake
description = Demonstrates menu local tasks.
package = Pro Drupal Development
core = 7.x
files[] = milkshake.module
```

Затем создайте файл `sites/all/modules/custom/milkshake/milkshake.module`:

```

<?php
/**
 * @file
 * Модуль для изучения системы меню Drupal, в частности, локальных задач.
 */
/**
 * Реализация hook_menu().
 */
function milkshake_menu() {
  $items['milkshake'] = array(
    'title' => 'Milkshake flavors',
    'access arguments' => TRUE,
    'page callback' => 'milkshake_overview',
    'type' => MENU_NORMAL_ITEM,
  );
  $items['milkshake/list'] = array(
    'title' => 'List flavors',
    'access arguments' => TRUE,
    'type' => MENU_DEFAULT_LOCAL_TASK,
    'weight' => 0,
  );
  $items['milkshake/add'] = array(
    'title' => 'Add flavor',
    'access arguments' => TRUE,
    'page callback' => 'milkshake_add',
    'type' => MENU_LOCAL_TASK,
    'weight' => 1,
  );
  $items['milkshake/list/fruity'] = array(
    'title' => 'Fruity flavors',
    'access arguments' => TRUE,
    'page callback' => 'milkshake_list',
    'page arguments' => array(2),          // Передается 'fruity'.
    'type' => MENU_LOCAL_TASK,
  );
  $items['milkshake/list/candy'] = array(
    'title' => 'Candy flavors',
    'access arguments' => TRUE,
    'page callback' => 'milkshake_list',
    'page arguments' => array(2),          // Передается 'candy'.
    'type' => MENU_LOCAL_TASK,
  );
  return $items;
}
function milkshake_overview() {
  $output = t('The following flavors are available...');
  // ... какой-то код
  return $output;
}
function milkshake_add() {
  return t('A handy form to add flavors might go here...');
}
function milkshake_list($type) {
  return t('List @type flavors', array('@type' => $type));
}

```

Интерфейс с вкладками показан на рис. 4.12.



Рис. 4.12. Локальные задачи и меню в виде вкладок

Скрытие существующих элементов меню

Существующие элементы меню можно скрыть, изменив их атрибут `hidden`. Предположим, что по какой-то причине необходимо удалить элемент меню `Create content` (Создать контент). Для этого воспользуемся уже знакомой функцией `hook_menu_link_alter()`:

```
/**
 * Реализация hook_menu_link_alter().
 */
function menufun_menu_link_alter(&$item) {
  // Скрытие ссылки Create content.
  if ($item['link_path'] == 'node/add') {
    $item['hidden'] = 1;
  }
}
```

Использование `menu.module`

Активация модуля меню Drupal предоставляет удобный пользовательский интерфейс для администрирования сайта и настройки существующих меню вроде меню навигации или главного меню, а также для добавления новых меню. При выполнении функции `menu_rebuild()` из файла `includes/menu.inc` структура данных, представляющая дерево меню, сохраняется в базе данных. Это происходит при активации или отключении модулей или других действиях, которые меняют компоновку дерева меню. Данные сохраняются в таблице `menu_router`, а информация о ссылках — в таблице `menu_links`.

В процессе создания ссылок для страницы Drupal сначала создает дерево на основе информации путей, полученной из реализаций хуков меню в модулях и хранящейся в таблице `menu_router`, а затем эта информация переопределяется информацией о меню из базы данных. Такое поведение позволяет использовать модуль `menu.module` для изменения родительского пункта, пути, заголовка и описания дерева меню — базовое дерево не меняется, а создаваемые данные просто переопределяют его.

На заметку! Тип элемента меню (такой как `ENU_CALLBACK` или `DEFAULT_LOCAL_TASK`) представлен в базе данных своим десятичным эквивалентом.

Модуль `menu.module` также добавляет раздел в форму ноды для оперативного добавления элементов меню из текущей статьи.

Распространенные ошибки

Вы только что реализовали в своем модуле хук меню, но обратные вызовы не срабатывают, меню не отображаются или все вообще не работает. Ниже перечислены основные моменты, которые следует проверить.

- Возможно, вы указали в ключе `access callback` функцию, которая возвращает `FALSE`?
- Возможно, вы забыли добавить строку `return $items;` в конце хука меню?
- Возможно, вы записали значение аргументов `access` или `page` в виде строки, а не массива?
- Очистили ли вы кэш меню и перестроили ли меню?
- Если вы хотите, чтобы элементы меню имели вид вкладок, и задали тип `MENU_LOCAL_TASK`, то назначили ли вы родительский элемент, у которого имеется обратный вызов страницы?
- Если вы работаете с локальными задачами, то имеется ли на странице не менее двух вкладок (только тогда они появляются)?

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Уметь отображать URL-адреса на функции в своем модуле либо в других модулях или `.inc`-файлах.
- Знать, как работает управление доступом.
- Знать, как работают обобщенные символы в путях.
- Уметь создавать страницы с вкладками (локальные задачи), которые отображаются на функции.
- Уметь программно изменять существующие элементы меню и ссылки.

Для дальнейшего чтения можно порекомендовать комментарии в файле `menu.inc`. См. также <http://api.drupal.org/?q=api/group/menu/7>.

ГЛАВА 5

Работа с базами данных

Работа Drupal зависит от базы данных. Контент, комментарии, таксономия, меню, пользователи, роли, права доступа и почти все остальное хранится в базе данных и используется в Drupal в качестве источника информации, необходимой для отображения контента на сайте и управления доступом к различным его аспектам. Внутри Drupal имеется облегченный уровень абстракции баз данных — между кодом и самой базой данных. Этот уровень абстракции устраняет значительную часть сложностей взаимодействия с базой данных и защищает Drupal от различий между механизмами разных СУБД. В данной главе вы узнаете, как работает уровень абстракции баз данных и как его использовать. Вы увидите, как модули могут изменять запросы. И, наконец, вы научитесь с помощью API-интерфейса схемы Drupal вставлять в файл `.install` модуля запросы, необходимые для создания и изменения таблиц базы данных.

Определение параметров базы данных

Drupal узнает, к какой базе данных подключаться и какие имя пользователя и пароль использовать для установления соединения с базой данных, из файла `settings.php` для конкретного сайта. Этот файл обычно находится по пути `sites/example.com/settings.php` или `sites/default/settings.php`. Код, который определяет подключение к базе данных, выглядит примерно так:

```
$databases = array (
  'default' =>
    array (
      'default' =>
        array (
          'driver' => 'mysql',
          'database' => 'имя_базы_данных',
          'username' => 'Имя_пользователя',
          'password' => 'пароль',
          'host' => 'localhost',
          'port' => '',
          'prefix' => '',
        ),
      ),
    ),
);
```

Этот пример предназначен для СУБД MySQL. Пользователи PostgreSQL вставят в начало строки подключения не `mysql`, а `pgsql`. Конечно, указанные здесь имя базы данных, пользовательское имя и пароль должны быть актуальными для вашей базы данных. Это полномочия для доступа к базе данных, а не к Drupal, и они задаются при создании учетной записи базы данных с помощью средств СУБД. Инсталлятор Drupal

спрашивает имя и пароль пользователя, чтобы поместить их в массив `$databases` в файле `settings.php`.

Если для сайта в качестве СУБД используется `sqlite`, то настройка несколько упрощается. В качестве драйвера следует указать `sqlite`, а база данных задается путем, включающим имя базы данных:

```
$databases['default']['default'] = array(
  'driver' => 'sqlite',
  'database' => '/path/to/databasefilename',
);
```

Уровень абстракции баз данных

Работу с API-интерфейсом абстракции баз данных можно полностью оценить, только попробовав отказаться от него. Были ли у вас проекты, где требовалось изменить СУБД, и вы целыми днями просматривали код, чтобы модифицировать в нем вызовы функций и запросы, специфичные для конкретной СУБД? Работа с уровнем абстракции позволяет не задумываться о различиях в именах функций для различных СУБД, а также о совместимости запросов с ANSI SQL — т.е. вам не придется писать различные запросы для различных СУБД. Например, вместо вызова `mysql_query()` или `pg_query()` в Drupal используется `db_query()`, который позволяет не обращать внимания в бизнес-логике на нюансы конкретной СУБД.

Уровень абстракции баз данных в Drupal 7 основан на библиотеке объектов данных PHP (PHP Data Object — PDO) и служит двум основным целям. Во-первых, код не привязывается ни к какой конкретной СУБД. Во-вторых, выполняется очистка вводимых пользователем данных, которые заносятся в запросы, чтобы избежать атак внедрением SQL. В основе этого уровня лежит принцип, что написание SQL удобнее, чем изучение языка работы с новой СУБД.

В Drupal также имеется API-интерфейс схемы, который позволяет обобщенным образом описать схему базы данных (т.е. таблицы и поля) и возложить на Drupal перевод этой схемы в конкретные детали используемой СУБД. Более подробно об этом пойдет речь при рассмотрении файлов `.install`.

Drupal определяет тип СУБД, к которой надо подключиться, с помощью данных из массива `$database` в файле `settings.php`. Например, если элемент `$databases['default']['default']['driver']` содержит значение `mysql`, то Drupal включит файл `includes/database.mysql.inc`. Если это значение равно `pgsql`, Drupal включит файл `includes/database.pgsql.inc`, а в случае `sqlite` будет включен файл `includes/database.sqlite.inc`. Этот механизм показан на рис. 5.1.

Если вы используете еще не поддерживаемую СУБД, можете написать собственный драйвер, реализовав функции-оболочки для своей СУБД. Подробнее об этом написано в разделе “Написание собственного драйвера СУБД” в конце данной главы.

Подключение к базе данных

Drupal автоматически устанавливает подключение к базе данных во время обычного процесса загрузки, поэтому беспокоиться об этом не надо.

Если вы работаете не в Drupal (например, пишете автономный PHP-сценарий, или у вас уже есть PHP-код, который необходимо включить в базу данных Drupal), можно применять следующий подход:

```
// Указание текущего каталога Drupal PHP.
chdir('/full/path/to/your/drupal/installation');
```

```
// Загрузка Drupal до этапа СУБД.
include_once('./includes/bootstrap.inc');
drupal_bootstrap(DRUPAL_BOOTSTRAP_DATABASE);

// Теперь можно выполнять запросы с помощью db_query().
$result = db_query('SELECT title FROM {node}');
...
```

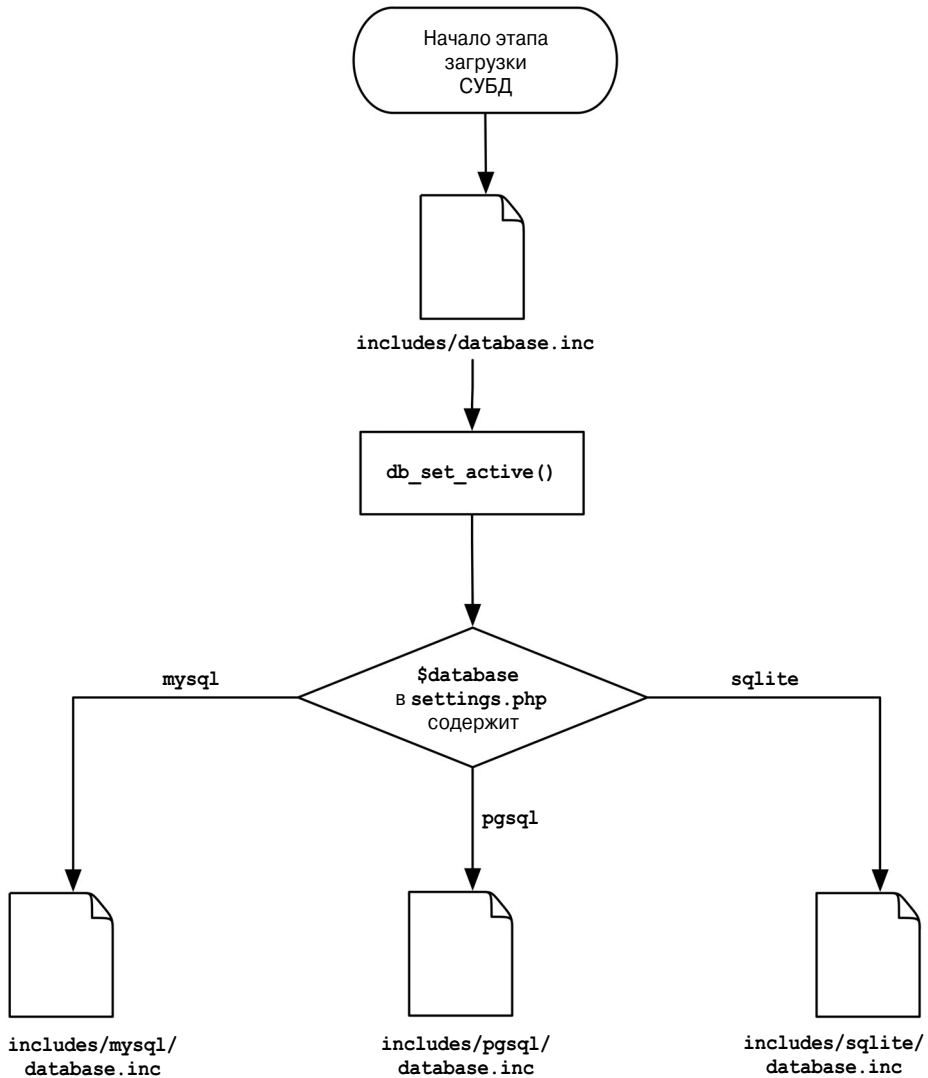


Рис. 5.1. Drupal определяет, какой файл СУБД необходимо включить, просматривая параметр `$database`

Внимание! Drupal часто настраивается так, что в каталоге `sites` содержатся несколько папок — тогда сайт можно будет перевести из среды разработки в промышленную эксплуатацию без изменения полномочий доступа к базе данных. Например, могут быть определены полномочия `sites/staging.example.com/settings.php` для тестового сервера СУБД и `sites/www.example.com/settings.php` для производственного сервера. При установлении подключения к базе данных, описанном в данном разделе, Drupal всегда использует `sites/default/settings.php`, т.к. при этом не нужно выполнять HTTP-запрос.

Выполнение простых запросов

Функция `Drupal db_query()` используется для выполнения запроса `SELECT` в активном подключении к базе данных. Имеются другие функции для выполнения запросов `INSERT`, `UPDATE` и `DELETE` — они будут описаны чуть ниже, а пока мы рассмотрим выборку информации из базы данных.

При написании SQL-запросов необходимо знать синтаксис, специфичный для Drupal. Первым делом, имена таблиц заключаются в фигурные скобки, чтобы при необходимости к ним можно было добавлять префиксы и получать уникальные имена. Это соглашение позволяет пользователям, у которых количество создаваемых баз данных ограничено их поставщиком хостинга, устанавливать Drupal в существующих базах данных и все-таки избежать конфликтов имен таблиц, указывая префиксы баз в файле `settings.php`. Вот пример простого запроса — выборки имени для роли 2:

```
$result = db_query('SELECT name FROM {role} WHERE rid = :rid', array(':rid' => 2));
```

Обратите внимание на применение `:rid` в качестве именованного заполнителя. В Drupal запросы всегда пишутся с заполнителями, а конкретные значения указываются в парах *ключ => значение*. Заполнитель `:rid` автоматически заменяется на значение, присвоенное ему в массиве, который используется для определения всех значений, назначенных заполнителям в запросе — в данном случае 2. Другие заполнители означают дополнительные параметры:

```
db_query('SELECT name FROM {role} WHERE rid > :rid AND rid < :max_rid',
  array(':rid' => 0, ':max_rid' => 3));
```

Когда дело дойдет до выполнения в базе данных, эта строка примет следующий вид:

```
SELECT FROM role WHERE rid > 0 and rid < 3
```

Данные, введенные пользователем, всегда следует передавать как отдельные параметры, чтобы можно было выполнить очистку этих значений и тем самым защититься от атак внедрением SQL.

Первым параметром функции `db_query()` всегда является сам запрос. Остальные параметры — это динамические значения для проверок и вставок в строку запроса. Эти значения передаются в виде массива пар *ключ => значение*.

Данный синтаксис переводит константы `TRUE`, `FALSE` и `NULL` в их десятичные эквиваленты (0 или 1). В большинстве случаев это не приводит к каким-либо проблемам.

Рассмотрим несколько примеров. В этих примерах мы используем таблицу базы данных по имени `joke`, которая содержит три поля: идентификатор ноды `nid` (целое число), идентификатор версии `vid` (целое число) и текстовое поле `punchline`, содержащее кульминационный момент шутки (дополнительные сведения относительно модуля `joke` ищите в главе 7).

Начнем с простого запроса. Нужно получить все строки со всеми полями из таблицы `joke`, в которых поле `vid` содержит значение `$node->vid`:

```
db_query('SELECT * FROM {joke} WHERE vid = :vid', array(':vid' => $node->vid));
```

Теперь вставим в таблицу `joke` новую строку с помощью функции `db_insert`. Вставляемые поля указываются конструкцией `->fields` и массивом пар *ключ => значение*, где ключи содержат имена полей, а значения — данные, вставляемые в поля новой строки. Обратите также внимание на конструкцию `->execute()` в конце оператора, которая как раз и выполняет вставку в базу данных.

```
$nid = db_insert('joke')
->fields(array(
  'nid' => '4',
  'vid' => 1,
  'punchline' => 'And the pig said oink!',
))
->execute();
```

Теперь обновим все строки в таблице `joke`: занесем кульминационный момент `'Take my wife please!'` во все строки, где `nid` больше или равен 3. Массив полей и значений для замены будет передан с помощью конструкции `->fields`, а условие для изменения значений — с помощью модификатора `->condition`:

```
$num_updated = db_update('joke')
->fields(array(
  'punchline' => 'Take my wife please!',
))
->condition('nid', 3, '>=')
->execute();
```

Если необходимо узнать количество строк, на которые повлияло обновление, после выполнения обновления можно воспользоваться значением, присвоенным переменной `$num_updated`.

А теперь удалим из таблицы `joke` все строки с кульминационным моментом `'Take my wife please!'`. Для этого понадобится функция `db_delete` с модификатором `->condition`, позволяющим указать условия удаления записей:

```
$num_deleted = db_delete('joke')
->condition('punchline', 'Take my wife please!')
->execute();
```

Выборка результатов запроса

Существует несколько способов выборки результатов запросов в зависимости от того, нужна одна строка или же набор результатов, диапазон результатов для внутреннего использования либо для отображения с разбивкой на страницы.

Получение одного значения

Если требуется получить из базы данных только одно значение, можно применять метод `->fetchField()`, который позволяет выбрать это значение. Вот пример выборки общего количества записей в таблице `joke`:

```
$nbr_records = db_query("SELECT count(nid) FROM {joke}")->fetchField();
```

Получение нескольких строк

В большинстве случаев из базы данных необходимо возвращать более одного поля. Вот типичный образец перебора строк в наборе результатов:

```

$type = 'page';
$status = 1;

$result = db_query(
  "SELECT nid, title FROM {node} WHERE type = :type AND status = :status",
  array(
    ':type' => $type, ':status' => 1,
  ));
foreach ($result as $row) {
  echo $row->title."<br/>";
}

```

Этот фрагмент кода выводит заголовки всех опубликованных нод, имеющих тип page (поле status в таблице node равно 0 для неопубликованных нод и 1 для опубликованных). Вызов db_query возвращает массив результатов, где каждый элемент представляет собой строку из таблицы, удовлетворяющую критерию, который указан в запросе. Цикл foreach позволяет перебрать элементы массива результатов и, как в данном случае, вывести заголовок каждой ноды в отдельной строке.

Использование конструктора запросов и объектов запросов

Одним из нововведений, появившихся в Drupal 7, является возможность построения объектов запросов с помощью конструктора запросов. В приведенных выше примерах запросы были относительно просты, но если понадобится написать более сложные запросы? И вот здесь может пригодиться конструктор запросов и объекты запросов. Сейчас мы рассмотрим один пример, а затем будет описана концепция на основе создания более сложных запросов.

В одном из приведенных выше примеров создавался запрос, который выбирал значения из таблицы, если идентификатор роли был больше или равен 2. Запрос имел следующий вид:

```
$result = db_query('SELECT name FROM {role} WHERE rid = :rid', array(':rid' => 2));
```

Теперь напишем этот же запрос с помощью объекта запроса и конструктора запросов. Вначале создадим объект запроса — для этого потребуется выбрать нужную таблицу и назначить ей идентификатор (r), чтобы в дальнейшем ссылаться на поля из этой таблицы:

```
$query = db_select('role', 'r');
```

Затем расширим запрос, указав условие (rid = 2) и поля для возврата из запроса:

```

$query
  ->condition('rid', 2)
  ->fields('r', array('name'));

```

После этого можно выполнить запрос и присвоить набор результатов переменной \$result:

```
$result = $query->execute();
```

Для вывода результатов необходимо пройти в цикле по элементам возвращенного массива:

```

foreach($result as $row) {
  echo $row->name."<br/>";
}

```

Использование объекта запроса и конструктора запросов облегчает создание сложных запросов к базам данных. В следующих примерах будет показано, как работать с конструктором запросов.

Получение ограниченного диапазона результатов

Выполнение запросов, которые могут вернуть сотни или даже тысячи записей — это риск, который стоит обдумать еще на этапе написания запроса. Один из механизмов минимизации риска — использование модификатора диапазона, который позволяет ограничить максимальное количество записей, возвращаемых запросом. Примером может служить запрос, который возвращает все ноды типа `page`. Если сайт содержит тысячи нод, запрос может выполняться длительное время и выдать огромный объем информации. С помощью модификатора диапазона можно ограничить количество строк, возвращаемых запросом, и устранить риск длительного выполнения и большого объема данных.

Следующий запрос добавляет модификатор диапазона с начальным номером записи 0 и конечным номером 100:

```
$query = db_select('node', 'n');

$query
->condition('type', 'page')
->fields('n', array('title'))
->range(0,100);

$result = $query->execute();

foreach($result as $row) {
  echo $row->title."<br/>";
}
```

Получение результатов для страничного вывода

Если запрос возвращает большое количество строк, можно воспользоваться разбивкой на страницы. Такая разбивка ограничивает количество строк, выводимых на одной странице, и обеспечивает навигационный элемент, который позволяет посетителю сайта перемещаться по страницам результатов. Примером может служить запрос, возвращающий 100 строк. Этот запрос можно настроить так, чтобы он выдавал по 10 строк и отображал кнопки “Вперед” для просмотра следующих 10 строк, “Назад” для просмотра предыдущих 10 строк, “В начало” для просмотра первых 10 строк и “В конец” для просмотра последних 10 строк, а также ссылки для щелчка на конкретном номере страницы и перехода на эту страницу результатов (например, щелчок на ссылке 5 приведет к просмотру строк 41–50).

Для демонстрации разбивки на страницы мы создадим запрос, который возвращает все страничные ноды в таблице нод и выводит результаты порциями по 10 строк на странице с панелью пейджера в нижней части.

Вначале мы создадим объект запроса и расширим его, указав Drupal создать объект запроса с пейджером:

```
$query = db_select('node', 'n')->extend('PagerDefault');
```

Потом с помощью модификатора-ограничителя добавим условие, поля и количество элементов, которые нужно выводить на странице:


```
$query
->condition('type', 'page')
->fields('n', array('title'))
->limit(10);
```

Затем выполним запрос и посмотрим набор результатов, добавляя каждую строку в выходную переменную с удобным именем `$output`:

```
$output = '';
foreach ($result as $row) {
    $output .= $row->title."<br/>";
}
```

После этого вызовем функцию применения к полученным данным темы с пейджером — тогда выходные данные будут выводиться по 10 элементов на странице с пейджером в нижней части этой страницы (рис. 5.2) — и отобразим результаты. Подробное описание, как пейджер работает с результатами запроса к базе данных и как уровень темы отображает постраничные результаты, можно прочитать в файле `/includes/pager.inc`.

```
$output .= theme('pager');
print $output;
```

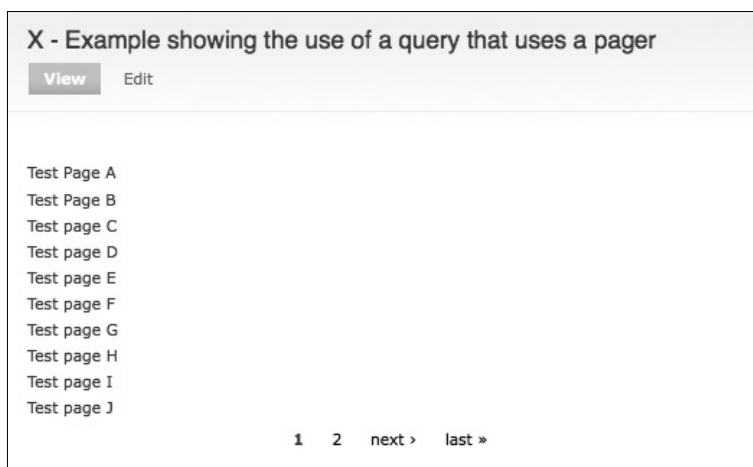


Рис. 5.2. Пейджер — встроенное средство Drupal для навигации по набору результатов

Другие распространенные запросы

Уровень баз данных Drupal 7 способен выполнять ряд других распространенных функций, которые могут потребоваться. Первый пример — сортировка набора результатов с помощью метода `orderBy`. В приведенном ниже примере набор результатов упорядочивается по возрастанию заголовков:

```
$query
->condition('type', 'page')
->fields('n', array('title'))
->orderBy('title', 'ASC');
```

Следующий пример выполняет сортировку по-другому — сначала по дате изменения ноды в убывающем порядке, а затем по заголовкам в порядке возрастания:

```
$query
->condition('type', 'page')
->fields('n', array('title', 'changed'))
->orderBy('changed', 'DESC')
->orderBy('title', 'ASC');
```

Могут существовать и запросы, приводящие к появлению повторяющихся результатов. В таком случае можно отфильтровать такие записи с помощью метода `distinct`:

```
$query
->condition('type', 'page')
->fields('n', array('title', 'changed'))
->orderBy('changed', 'DESC')
->orderBy('title', 'ASC')
->distinct();
```

Дополнительную информацию и примеры можно найти на странице <http://drupal.org/node/310069>.

Вставки и обновления данных с помощью `drupal_write_record()`

Вставки в базу данных новых строк и обновления существующих строк являются распространенной программистской задачей. Код обычно определяет, что нужно осуществить — вставку или изменение — а затем выполняет необходимую операцию.

Поскольку каждая таблица, используемая в Drupal, описывается с помощью схемы, Drupal знает, какие поля содержатся в таблицах и каковы значения по умолчанию для каждого поля. Передав ассоциативный массив полей и значений в функцию `drupal_write_record()`, можно поручить Drupal сгенерировать и выполнить SQL-запрос, не делая этого вручную.

Предположим, что имеется таблица с записями о коллекции гигантских кроликов. Хук схемы для модуля с описанием таблицы выглядит так:

```
/**
 * Реализация hook_schema().
 */
function bunny_schema() {
  $schema['bunnies'] = array(
    'description' => t('Stores information about giant rabbits.'),
    // Хранит информацию о гигантских кроликах.
    'fields' => array(
      'bid' => array(
        'type' => 'serial',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => t('Primary key: A unique ID for each bunny.'),
        // Первичный ключ: уникальный идентификатор для каждого кролика.
      ),
      'name' => array(
        'type' => 'varchar',
        'length' => 64,
        'not null' => TRUE,
        'description' => t('Each bunny gets a name.'),
        // У каждого кролика есть имя.
      ),
      'tons' => array(
```

```

    'type' => 'int',
    'unsigned' => TRUE,
    'not null' => TRUE,
    'description' => t('The weight of the bunny to the nearest ton.'),
    // Вес кролика с точностью до тонны.
  ),
),
'primary key' => array('bid'),
'indexes' => array(
  'tons' => array('tons'),
),
);
return $schema;
}

```

Вставка новых записей и обновление существующих выполняются элементарно:

```

$table = 'bunnies';
$record = new stdClass();
$record->name = t('Bortha');
$record->tons = 2;
drupal_write_record($table, $record);

// Новый идентификатор кролика $record->bid устанавливается с помощью
// drupal_write_record(), т.к. $record передается по ссылке.
watchdog('bunny', 'Added bunny with id %id.', array('%id' => $record->bid));

// Придумали другое имя.
$record->name = t('Bertha');

// Теперь изменим запись в базе данных.
// Для изменения нужно передать имя первичного ключа таблицы.
drupal_write_record($table, $record, 'bid');

watchdog('bunny', 'Updated bunny with id %id.', array('%id' => $record->bid));

```

Поддерживается и синтаксис с массивами, хотя если \$record является массивом, то drupal_write_record() преобразует его во внутренний объект.

API-интерфейс схемы

С помощью уровня абстракции баз данных Drupal поддерживает различные СУБД — MySQL, PostgreSQL, SQLite и т.д. Каждый модуль, которому для работы нужна таблица базы данных, описывает эту таблицу для Drupal с помощью определения схемы. Затем Drupal транслирует такое определение в синтаксис, понятный для конкретной СУБД.

Использование файлов .install

Как было сказано в главе 2, при написании модуля, в котором нужно создавать и сохранять одну или несколько таблиц баз данных, инструкции по созданию и обслуживанию табличной структуры заносятся в файл .install, который распространяется вместе с модулем.

Создание таблиц

Во время инсталляции нового модуля Drupal автоматически проверяет, имеется ли в файле .install определение схемы (рис. 5.3). Если имеется, то Drupal создает табли-

цу (или таблицы) базы данных в этой схеме. Ниже приведен пример общей структуры определения схемы:

```
$schema['tablename'] = array(
  // Описание таблицы.
  'description' => t('Description of what the table is used for.'),
                        // Описание, для чего предназначена таблица.
  'fields' => array(
    // Определение поля.
    'field1' => array(
      'type' => 'int',
      'unsigned' => TRUE,
      'not null' => TRUE,
      'default' => 0,
      'description' => t('Description of what this field is used for.'),
                        // Описание, для чего предназначено поле.
    ),
  ),
  // Объявления индексов.
  'primary key' => array('field1'),
);
```

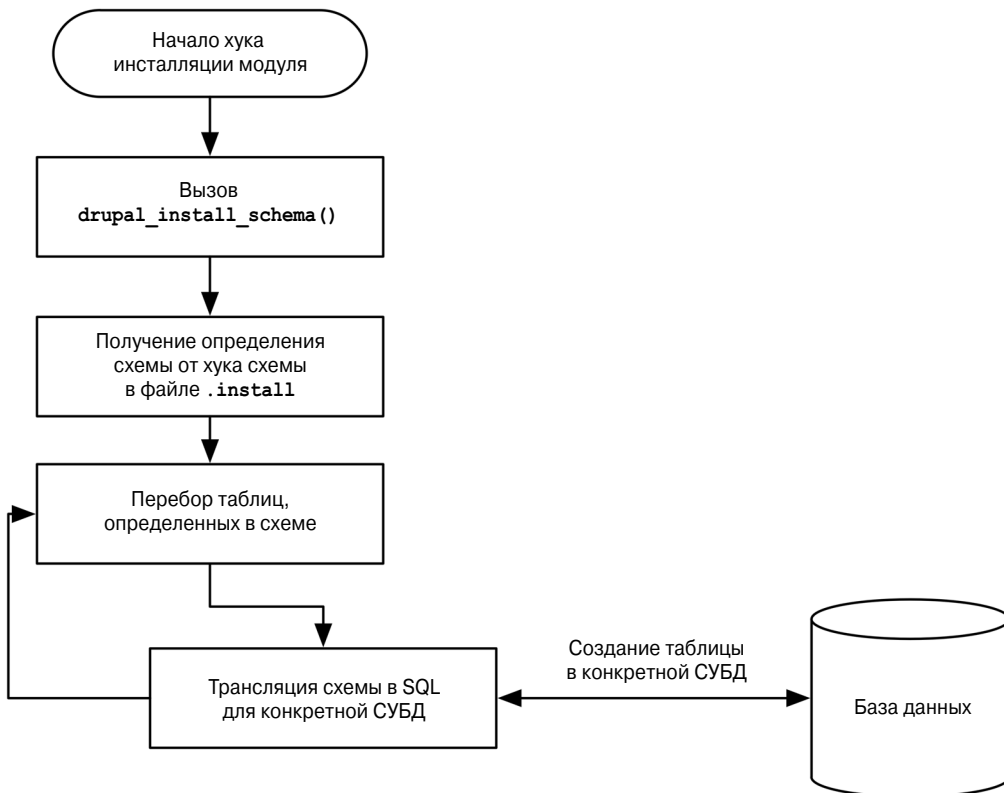


Рис. 5.3. Использование определения схемы для создания таблиц базы данных

Рассмотрим определение схемы для таблицы книг по Drupal, которая находится по адресу `modules/book/book.install`:

```
/**
 * Реализация hook_schema().
 */
function book_schema() {
  $schema['book'] = array(
    'description' => 'Stores book outline information. Uniquely connects each node
                      in the outline to a link in {menu_links}',
    // Хранит краткие описания книг. Уникально сочетает каждую ноду
    // в кратких описаниях со ссылкой в {menu_links}
    'fields' => array(
      'mlid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => "The book page's {menu_links}.mlid.",
        // Идентификатор ссылки меню для страницы книги
      ),
      'nid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => "The book page's {node}.nid.",
        // Идентификатор ноды для страницы книги
      ),
      'bid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => "The book ID is the {book}.nid of the top-level page.",
        // Идентификатор книги — это идентификатор ноды страницы верхнего уровня.
      ),
    ),
    'primary key' => array('mlid'),
    'unique keys' => array(
      'nid' => array('nid'),
    ),
    'indexes' => array(
      'bid' => array('bid'),
    ),
  );

  return $schema;
}
```

Это определение схемы описывает таблицу `book` с тремя полями типа `int`. В нем также указаны первичный ключ, уникальный индекс (т.е. все значения в данном поле уникальны) и обычный индекс. Обратите внимание, что когда описание поля ссылается на поле из другой таблицы, используются фигурные скобки. Это позволяет модулю схемы (см. следующий раздел) создать удобные гиперссылки на описания таблиц.

Использование модуля схемы

Вот тут вы можете подумать: “Ничего себе! Создание таких больших массивов для описания в Drupal всех моих таблиц — это жуткая рутина!” Не пугайтесь. Просто загрузите модуль схемы из <http://drupal.org/project/schema> и активируйте его для своего сайта. Выберите пункт меню Structure⇒Schema (Структура⇒Схема) и перейдите на вкладку Inspect (Просмотр) — это позволит увидеть определение схемы для любой таблицы базы данных. Так что если вы создали таблицу с помощью SQL, вы сможете, воспользовавшись модулем схемы, получить для нее определение схемы, а затем скопировать и вставить его в файл `.install`.

Совет. Написание схемы с нуля почти никогда не требуется. Вместо этого используйте существующую таблицу (таблицы) и вкладку Inspect модуля схемы, чтобы автоматически построить схему. Можно воспользоваться и другими средствами, такими как модуль мастера таблиц (Table Wizard — <http://drupal.org/project/tw>), чтобы просматривать всю информацию о любой таблице в Drupal с помощью модуля Views.

Модуль схемы позволяет также просмотреть схему любого модуля. Например, на рис. 5.4 показано представление модуля схемы для книг. Обратите внимание на то, как имена таблиц, заключенные в фигурные скобки в описаниях таблиц и полей, превращаются в удобные ссылки.

```
▼ BOOK
$schema['book'] = array(
  'description' => 'Stores book outline information. Uniquely connects each...',
  'fields' => array(
    'm1id' => array(
      'description' => 'The book page's menu_links.m1id.',
      'type' => 'int',
      'unsigned' => TRUE,
      'not null' => TRUE,
      'default' => 0,
    ),
    'nid' => array(
```

Рис. 5.4. Модуль схемы выводит схему модуля книг

Отображение типов полей из схемы в базу данных

Тип поля, объявленный в определении схемы, отображается на внутренний тип поля в СУБД. Например, целочисленное поле с объявленным размером `tiny` становится полем `TINYINT` в MySQL или полем `small int` в PostgreSQL. Действительное отображение можно просмотреть, отобразив результаты функции `getFieldTypeMap()`, или обратившись к табл. 5.1 далее в главе.

Текстовые типы

Текстовые поля содержат текст.

Тип *varchar*

Поля символов переменной длины — наиболее часто применяемый тип поля для хранения текста длиной до 256 символов. Максимальное количество символов задается в ключе `length`. Длина поля `varchar` может быть 0–255 символов в MySQL 5.0.2 и более старых версиях, и 0–65535 в MySQL 5.0.3 и более новых версиях. Длины полей `varchar` в PostgreSQL могут быть еще больше.

```
$field['fieldname'] = array(
    'type' => 'varchar',           // Обязательно.
    'length' => 255,               // Обязательно.
    'not null' => TRUE,            // По умолчанию FALSE.
    'default' => 'chocolate',     // См. ниже.
    'description' => t('Always state the purpose of your field.'),
    // Всегда описывайте назначение поля.
);
```

Если ключ `default` не задан, а ключ `not null` установлен в `FALSE`, то в качестве значения по умолчанию принимается `NULL`.

Тип *char*

Поля типа `char` являются символьными полями фиксированной длины. Количество символов определяется ключом `length`. Длины полей `char` в MySQL могут быть от 0 до 255 символов.

```
$field['fieldname'] = array(
    'type' => 'char',             // Обязательно.
    'length' => 64,              // Обязательно.
    'not null' => TRUE,          // По умолчанию FALSE.
    'default' => 'strawberry',   // См. ниже.
    'description' => t('Always state the purpose of your field.'),
);
```

Если ключ `default` не задан, а ключ `not null` установлен в `FALSE`, то в качестве значения по умолчанию принимается `NULL`.

Тип *text*

Поля типа `text` используются для текстовых данных, которые могут иметь большой размер. Например, поле `body` из таблицы `node_revisions` (где хранится текст тела ноды) — как раз такого типа. Для полей типа `text` значения по умолчанию не могут использоваться.

```
$field['fieldname'] = array(
    'type' => 'text',             // Обязательно.
    'size' => 'small',           // tiny | small | normal | medium | big
    'not null' => TRUE,          // По умолчанию FALSE.
    'description' => t('Always state the purpose of your field.'),
);
```

Числовые типы

Числовые типы данных используются для хранения чисел и могут быть `integer`, `serial`, `float` и `numeric`.

Тип *int*

Этот тип поля используется для хранения целых чисел, такие как идентификаторы нод. Если ключ `unsigned` установлен в `TRUE`, то отрицательные значения не допускаются.

```
$field['fieldname'] = array(
    'type' => 'int',             // Обязательно.
    'unsigned' => TRUE,         // По умолчанию FALSE.
    'size' => 'small',          // tiny | small | medium | normal | big
    'not null' => TRUE,         // По умолчанию FALSE.
    'description' => t('Always state the purpose of your field.'),
);
```

Тип *serial*

Поле типа *serial* хранит автоинкрементные числа. Например, при добавлении ноды значение поля *nid* в таблице *node* увеличивается на 1. Для этого нужно вставить строку и вызвать функцию `db_last_insert_id()`. Если строка добавлена в другом потоке между вставкой строки и выборкой последнего идентификатора, то все равно возвращается корректный идентификатор, поскольку он отслеживается для каждого подключения отдельно. Автоинкрементные поля должны быть индексированными; обычно они служат первичными ключами.

```
$field['fieldname'] = array(
  'type' => 'serial', // Обязательно.
  'unsigned' => TRUE, // По умолчанию FALSE. Серийные числа обычно положительны.
  'size' => 'small', // tiny | small | medium | normal | big
  'not null' => TRUE, // По умолчанию FALSE. Обычно TRUE
                      // для автоинкрементных полей.
  'description' => t('Always state the purpose of your field.'),
);
```

Тип *float*

Числа с плавающей точкой хранятся в типе данных *float*. Обычно нет особой разницы между числами размеров *tiny*, *small*, *medium* и *normal*; размер *big* означает поле с двойной точностью.

```
$field['fieldname'] = array(
  'type' => 'float', // Обязательно.
  'unsigned' => TRUE, // По умолчанию FALSE.
  'size' => 'normal', // tiny | small | medium | normal | big
  'not null' => TRUE, // По умолчанию FALSE.
  'description' => t('Always state the purpose of your field.'),
);
```

Тип *numeric*

Тип *numeric* позволяет задавать точность и масштаб чисел. Точность — это общее количество значащих цифр в числе, а масштаб — общее количество цифр справа от десятичной точки. Например, у числа 123.45 точность 5 и масштаб 2. Ключ *size* не используется. На момент написания этих строк поля *numeric* в схеме ядра Drupal не использовались.

```
$field['fieldname'] = array(
  'type' => 'numeric', // Обязательно.
  'unsigned' => TRUE, // По умолчанию FALSE.
  'precision' => 5, // Значащие цифры.
  'scale' => 2, // Цифры справа от десятичной точки.
  'not null' => TRUE, // По умолчанию FALSE.
  'description' => t('Always state the purpose of your field.'),
);
```

Дата и время: тип *datetime*

Этот тип данных не используется в ядре Drupal, вместо него применяются метки времени Unix в целочисленных полях. Формат *datetime* является составным: он содержит одновременно дату и время.

```
$field['fieldname'] = array(
  'type' => 'datetime', // Обязательно.
  'not null' => TRUE, // По умолчанию FALSE.
  'description' => t('Always state the purpose of your field.'),
);
```


Двоичный: тип *blob*

Тип данных больших двоичных объектов (binary large object data — blob) применяется для хранения двоичных данных (например, таблица кэша Drupal используется для хранения кэшированных данных). Двоичные данные могут содержать музыку, изображения или видео. Доступны два размера: *normal* (нормальный) и *big* (большой).

```
$field['fieldname'] = array(
  'type' => 'blob',           // Обязательно.
  'size' => 'normal',         // normal | big
  'not null' => TRUE,         // По умолчанию FALSE.
  'description' => t('Always state the purpose of your field.'),
);
```

Объявление конкретного типа столбца с помощью *mysql_type*

Если тип столбца в используемой СУБД в точности известен, то в определении схемы можно установить ключ *mysql_type* (или *pgsql_type*). Он переопределяет ключи типа и размера для этой СУБД. Например, в MySQL имеется тип поля *TINYBLOB* для небольших двоичных объектов. Чтобы указать, что среда Drupal должна использовать *TINYBLOB* при работе с MySQL, но вернуться к обычному типу *BLOB* в случае другой СУБД, поле понадобится определить следующим образом:

```
$field['fieldname'] = array(
  'mysql_type' > 'TINYBLOB', // Для использования с MySQL.
  'type' => 'blob',          // Для других СУБД.
  'size' => 'normal',        // Для других СУБД.
  'not null' => TRUE,
  'description' => t('Wee little blobs.')
);
```

Встроенные типы MySQL и PostgreSQL приведены в табл. 5.1.

Таблица 5.1. Отображение ключей *type* и *size* в определениях схемы на типы конкретных СУБД

Определение схемы		Встроенный тип поля в СУБД		
Тип (type)	Размер (size)	MySQL	PostgreSQL	SQLite
varchar	normal	VARCHAR	varchar	VARCHAR
char	normal	CHAR	character	VARCHAR
text	tiny	TINYTEXT	text	TEXT
text	small	TINYTEXT	text	TEXT
text	medium	MEDIUMTEXT	text	TEXT
text	big	LONGTEXT	text	TEXT
text	normal	TEXT	text	TEXT
serial	tiny	TINYINT	serial	INTEGER
serial	small	SMALLINT	serial	INTEGER
serial	medium	MEDIUMINT	serial	INTEGER
serial	big	BIGINT	bigserial	INTEGER
serial	normal	INT	serial	INTEGER
int	tiny	TINYINT	smallint	INTEGER

Определение схемы		Встроенный тип поля в СУБД		
Тип (type)	Размер (size)	MySQL	PostgreSQL	SQLite
int	small	SMALLINT	smallint	INTEGER
int	medium	MEDIUMINT	int	INTEGER
int	big	BIGINT	bigint	INTEGER
int	normal	INT	int	INTEGER
float	tiny	FLOAT	real	FLOAT
float	small	FLOAT	real	FLOAT
float	medium	FLOAT	real	FLOAT
float	big	DOUBLE	double precision	FLOAT
float	normal	FLOAT	real	FLOAT
numeric	normal	DECIMAL	numeric	NUMERIC
blob	big	LONGBLOB	bytea	BLOB
blob	normal	BLOB	bytea	BLOB
datetime	normal	DATETIME	timestamp	TIMESTAMP

Обслуживание таблиц

При создании новой версии модуля иногда требуется изменить схему базы данных. Возможно, необходимо добавить столбец или индекс для столбца. Просто взять и создать новую таблицу нельзя, т.к. она содержит данные. Вот так можно изменить базу без ущерба для данных.

1. Измените реализацию `hook_schema()` в файле `.install`, чтобы у новых пользователей, которые будут устанавливать модуль, устанавливалась новая схема. Определение схемы в файле `.install` всегда будет самой свежей схемой для таблиц и полей модуля.
2. Предоставьте существующим пользователям путь обновления, написав для этого функцию обновления. Функции обновления должны иметь последовательные имена, начиная с номера версии Drupal. Например, первая функция обновления для Drupal 7 должна называться `modulename_update_7000()`, а вторая — `modulename_update_7001()`. Ниже приведен пример из файла `modules/comment/comment.install`, где таблица, используемая для хранения комментариев, переименована с `comments` в `comment`:

```
/**
 * Переименование таблицы {comments} в {comment}.
 */
function comment_update_7002() {
  db_rename_table('comments', 'comment');
}
```

3. Эта функция будет выполняться, когда пользователь запустит сценарий `http://example.com/update.php` после обновления модуля.

Внимание! Поскольку определение схемы в реализации `hook_schema()` изменяется при каждом добавлении таблицы, поля или индекса, функции обновления не должны использовать это определение схемы. Реализацию `hook_schema()` следует считать как находящуюся в настоящем, а функции обновления — в прошлом. См. <http://drupal.org/node/150220>.

Полный список функций для работы со схемами приведен на странице <http://api.drupal.org/api/group/schemaapi/7>.

Совет. Среда Drupal следит за тем, какую версию схемы использует модуль в данный момент. Эта информация находится в таблице `system`. После выполнения обновления, приведенного в данном разделе, в строке для модуля комментария будет содержаться значение `schema_version`, равное 7002. Чтобы Drupal отбросила это значение, используйте параметр `Reinstall Modules` в модуле `Devel` или удалите строку модуля из таблицы `system`.

Удаление таблиц при деинсталляции

При отключении модуля все данные, сохраненные этим модулем в базе данных, остаются без изменений, на случай, если администратор передумает и снова активирует модуль. На странице `Modules (Модули)` имеется вкладка `Uninstall (Деинсталляция)`, которая автоматически удаляет данные из базы. Может понадобиться удалить все переменные, определенные в то же время. Вот пример для модуля аннотирования, который был написан в главе 2:

```
/**
 * Реализация hook_uninstall().
 */
function annotate_uninstall() {
  // Очистка записи в таблице переменных.
  variable_del('annotate_nodetypes');
}
```

Изменения существующих схем с помощью `hook_schema_alter()`

Обычно модули сами создают и используют свои таблицы. Но что если модулю надо изменить существующую таблицу? Допустим, модулю абсолютно необходимо добавить столбец в таблицу `node`. Конечно, проще всего взять базу данных и добавить нужный столбец. Но тогда перестанут быть актуальными определения схемы Drupal, которые должны соответствовать реальной таблице базы данных. Существует лучший способ: `hook_schema_alter()`.

Предположим, что имеется модуль, который каким-нибудь образом помечает ноды, и по соображениям производительности нужно обязательно использовать существующую таблицу `node`, а не собственную таблицу, соединенную с идентификаторами нод. Модуль должен сделать две вещи: изменить таблицу `node` при инсталляции модуля и изменить схему, чтобы она соответствовала структуре базы данных. Первый пункт выполняется с помощью функции `hook_install()`, а второй — с помощью `hook_schema_alter()`. Если модуль называется `markednode.module`, то файл `markednode.install` будет содержать следующие инструкции:

```
/**
 * Реализация hook_install().
 */
function markednode_install() {
  $field = array(
    'type' => 'int',
    'unsigned' => TRUE,
    'not null' => TRUE,
    'default' => 0,
    'initial' => 0, // Задаёт начальное значение для существовавших ранее нод.
    'description' => t('Whether the node has been marked by the markednode module.'),
    // Помечена ли нода модулем markednode
  );
}
```

```

// Создание обычного индекса 'marked' по полю 'marked'.
$keys['indexes'] = array(
    'marked' => array('marked')
);

db_add_field('node', 'marked', $field, $keys);
}

/**
 * Реализация hook_schema_alter(). Содержимое $schema изменяется по ссылке.
 *
 * @param $schema
 *   Схема для всей системы, собранная drupal_get_schema().
 */
function markednode_schema_alter(&$schema) {
    // Добавление поля в существующую схему.
    $schema['node']['fields']['marked'] = array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => t('Whether the node has been marked by the markednode module.'),
        // Помечена ли нода модулем markednode
    );
}

```

Изменение запросов других модулей с помощью hook_query_alter()

Этот хук используется для изменения запросов, созданных где угодно в Drupal, без непосредственного изменения кода самих модулей. Все объекты запроса динамической выборки передаются через функцию `hook_query_alter()` методом `execute()` непосредственно перед компиляцией строки запроса. Это предоставляет модулям возможность произвольной обработки запроса. Функция `hook_query_alter()` принимает единственный параметр — сам объект запроса выборки.

В качестве примера работы `hook_query_alter()` в модуле `dbtest` выполняется изменение двух запросов. Первое изменение происходит, когда обнаруживается запрос с тегом `db_test_alter_add_range`. В этом случае в запрос добавляется диапазон (0, 2). Второе изменение выполняется при обнаружении запроса с тегом `db_test_alter_add_join`. В этом случае выполняется соединение таблиц `test` и `people`.

На заметку! Теги (tag) — это строки, идентифицирующие запрос. У запроса может быть любое количество тегов. Теги используются для пометки запросов, чтобы хуки изменения могли определить, нужно ли им срабатывать. Они должны содержать только строчные буквы, цифры и подчеркивания, причем начинаться обязательно с буквы — т.е. удовлетворять правилам создания идентификаторов в PHP.

```

function dbtest_query_alter(SelectQuery $query) {
    // Можно добавить диапазон.
    if ($query->hasTag('db_test_alter_add_range')) {
        $query->range(0, 2);
    }
}

```

```
// Или добавить соединение.
if ($query->hasTag('db_test_alter_add_join')) {
    $people_alias = $query->join('test', 'people', "test_task.pid=people.id");
    $name_field = $query->addField('name', 'people', 'name');
    $query->condition($people_alias . '.id', 2);
}
...
?>
```

Подключение к нескольким базам данных в Drupal

Уровень абстракции баз данных облегчает запоминание имен функций, а, кроме того, еще и обеспечивает защиту запросов. Иногда бывает нужно подключиться к сторонним или унаследованным базам данных, и было бы хорошо использовать для этого — а также для защиты запросов — API-интерфейс баз данных Drupal. И это действительно возможно! Например, модуль может открыть подключение к базе данных вне Drupal и извлечь оттуда данные.

В файле `settings.php` имеется массив `$databases`, составленный из нескольких строк подключения к базам данных. Вот стандартный синтаксис для описания одного подключения:

```
array(
  'driver' => 'mysql',
  'database' => 'databasename',
  'username' => 'username',
  'password' => 'password',
  'host' => 'localhost',
  'port' => 3306,
  'prefix' => 'myprefix_',
);
```

Предположим, что имеются две базы данных: стандартная (с именем D7) и унаследованная, которая определена ниже:

```
$databases = array (
  'default' =>
    array (
      'default' =>
        array (
          'driver' => 'mysql',
          'database' => 'd7',
          'username' => 'username',
          'password' => 'userpassword',
          'host' => 'localhost',
          'port' => '',
          'prefix' => '',
        ),
      ),
    ),
  'legacy' =>
    array (
      'default' =>
        array (
          'driver' => 'mysql',
          'database' => 'legacydatabase',
```

```

    'username' => 'legacyusername',
    'password' => 'legacyuserpassword',
    'host' => '122.185.22.1',
    'port' => '6060',
  ),
),
);

```

На заметку! База данных, которая используется для работы сайта Drupal, всегда должна быть помечена как default.

Если нужно подключиться к одной из других баз данных в Drupal, ее можно активировать по ключевому имени, а потом вернуться к стандартному подключению:

```

// Получение информации из базы данных вне Drupal.
db_set_active('legacy');
$result = db_query("SELECT * FROM ldap_user WHERE uid = %d", $user->uid);

// Возврат к стандартному подключению.
db_set_active('default');

```

Внимание! Если переключиться на другое подключение к базе данных и попытаться выполнить что-нибудь вроде `t("текст")`, возникнет ошибка. Функция `t()` требует выполнения действий с базой данных, а подключение к базе данных остается переключенным даже за пределами области действия кода, где было выполнено это переключение. Так что всегда как можно скорее возвращайте подключение к базе данных в default, и внимательно следите, чтобы не вызывать код, который обращается к базе данных.

Поскольку уровень абстракции баз данных спроектирован для применения одинаковых имен функций для каждой базы данных, невозможно одновременно использовать несколько разных СУБД (например, MySQL и PostgreSQL). Однако в статье <http://drupal.org/node/19522> имеется описание, как выполнить одновременное подключение к MySQL и PostgreSQL из одного и того же сайта.

Использование временной таблицы

При выполнении больших объемов обработки данных бывает необходимо создать временную таблицу для работы запроса. Это можно сделать с помощью вызова `db_query_temporary()` такого вида:

```
$tablename = db_query_temporary($query, $arguments, $options);
```

Здесь `$query` — подготовленный к выполнению оператор запроса, `$arguments` — массив значений, которые подставляются в запрос, а `$options` — массив параметров для управления работой запроса. Возвращается имя временной таблицы.

После этого можно выполнять запросы к временной таблице, используя полученное имя:

```
$final_result = db_query('SELECT foo FROM '.$tablename);
```

Обратите внимание, что во временных таблицах не нужны фигурные скобки для добавления префиксов, т.к. временная таблица существует недолго и не подвергается процессу добавления префиксов. Однако имена постоянных таблиц всегда заключаются в фигурные скобки.

На заметку! Временные таблицы не применяются в ядре Drupal, и пользователь, от имени которого Drupal подключается к базе данных, может не иметь права на создание временных таблиц. Поэтому авторы модулей не должны предполагать, что все работающие в Drupal обладают такими правами.

Написание собственного драйвера СУБД

Предположим, что вам понадобилось написать уровень абстракции для новой, футуристической СУБД с названием DNABase, в которой для повышения производительности используются молекулярные вычисления. Можно не начинать все с нуля, а скопировать существующий уровень абстракции — например, реализацию PostgreSQL — и изменить его.

Сначала создадим копию файла `includes/database/pgsql/database.inc` с именем `includes/database/dnabase/database.inc`. Затем изменим логику внутри каждой функции-оболочки для отображения возможностей DNABase, а не PostgreSQL.

Протестировать систему можно, подключившись к базе данных DNABase в Drupal за счет изменения содержимого `$databases` в файле `settings.php`.

Подробная информация по написанию собственного драйвера СУБД доступна по адресу <http://drupal.org/node/310087>.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Понимать, что такое уровень абстракции баз данных.
- Составлять простые запросы.
- Получать единичные и множественные результаты из базы данных.
- Получать ограниченный диапазон результатов.
- Уметь использовать пейджер.
- Разбираться в API-интерфейсе схемы Drupal.
- Уметь писать запросы так, чтобы другие разработчики могли изменять их.
- Уметь изменять запросы из других модулей.
- Уметь подключаться к нескольким базам данных, в том числе и к унаследованным.
- Уметь писать свой драйвер уровня абстракции.

ГЛАВА 6

Работа с пользователями

Пользователи — это смысл работы с Drupal. Среда Drupal может помочь пользователям создавать общие проекты, совместно работать над ними и формировать онлайн-сообщества. В данной главе вы узнаете о механизмах аутентификации, входа и внутреннего представления пользователей. Мы начнем с рассмотрения объекта `$user` и его структуры. Затем разберемся с процессом регистрации, входа и аутентификации пользователей. В конце главы будет показано, как связать Drupal с внешними системами аутентификации, такими как LDAP и Pubcookie.

Объект `$user`

Согласно требованиям Drupal, пользователь для выполнения входа должен иметь включенные cookie-наборы: пользователь с отключенными cookie-наборами может взаимодействовать с Drupal только как *анонимный пользователь*.

Во время процесса загрузки Drupal создает глобальный объект `$user`, который представляет сущность текущего пользователя. Если пользователь не вошел в систему (и поэтому не имеет cookie-набора сеанса), он считается анонимным пользователем. Код создания анонимного пользователя выглядит следующим образом (и находится в файле `includes/bootstrap.inc`):

```
function drupal_anonymous_user($session = '') {
  $user = new stdClass();
  $user->uid = 0;
  $user->hostname = ip_address();
  $user->roles = array();
  $user->roles[DRUPAL_ANONYMOUS_RID] = 'anonymous user';
  $user->session = $session;
  $user->cache = 0;
  return $user;
}
```

С другой стороны, если пользователь выполнил вход, объект `$user` создается соединением таблиц `users`, `user_roles` и `sessions` по идентификатору пользователя. Значения всех полей из всех таблиц заносятся в объект `$user`.

На заметку! Идентификатор пользователя — это целое число, которое назначается, когда пользователь регистрируется или администратор создает пользовательскую учетную запись. Этот идентификатор является первичным ключом в таблице `users`.

Содержимое объекта `$user` легко просмотреть: для этого нужно добавить в файл `index.php` операторы `global $user; print_r($user);`. Ниже показано, как обычно выглядит объект `$user` для выполнившего вход пользователя:

```
stdClass Object (
    [uid] => 1
    [name] => admin
    [pass] => $$CnUvfOYdOx1/Usy.X/Y9/SCmOLLY6QldrZjf7EOW0fR4LG7rCAmR
    [mail] => joe@example.com
    [theme] =>
    [signature] =>
    [signature_format] => 0
    [created] => 1277957059
    [access] => 1278254230
    [login] => 1277990573
    [status] => 1
    [timezone] =>
    [language] =>
    [picture] => 0
    [init] => joe@example.com
    [data] =>
    [sid] => 8cnG9e0jsCC7I7IYwfWB0rmRozIbaLlk35IQGN5fz9k
    [ssid] =>
    [hostname] => ::1
    [timestamp] => 1278254231
    [cache] => 0
    [session] => batches|a:1:{i:3;b:1;}
    [roles] => Array (
        [2] => authenticated user
        [3] => administrator
    )
)
```

В этом объекте `$user` данные полей, имена которых выделены курсивом, берутся из таблицы `sessions`. Назначение компонентов объекта `$user` объясняется в табл. 6.1.

Таблица 6.1. Компоненты объекта `$user`

Компонент	Описание
Из таблицы <i>users</i>	
<code>uid</code>	Идентификатор данного пользователя. Является первичным ключом таблицы <code>users</code> и уникален для данной инсталляции Drupal
<code>name</code>	Имя пользователя, вводимое пользователем при входе
<code>pass</code>	Хеш пароля пользователя, вычисленный по алгоритму sha512; сравнивается при входе пользователя. Сами пароли не сохраняются, и поэтому их невозможно восстановить, а можно только изменить
<code>mail</code>	Адрес электронной почты пользователя
<code>theme</code>	Поле не рекомендуется для использования, но оставлено по соображениям совместимости
<code>signature</code>	Подпись, которую пользователь вводит на странице своей учетной записи. Применяется, когда пользователь добавляет комментарий, и отображается, если только активирован модуль комментирования
<code>signature format</code>	Формат подписи пользователя (т.е. фильтрованный текст или полный текст)

Компонент	Описание
created	Метка времени Unix для момента создания учетной записи пользователя
access	Метка времени Unix для момента последнего обращения пользователя
login	Метка времени Unix для момента последнего успешного входа
status	Содержит 1, если пользователь на хорошем счету, и 0, если пользователь заблокирован
timezone	Количество секунд, на которое местное время пользователя отличается от GMT
language	Стандартный язык пользователя. Обычно пусто, за исключением случаев, когда на сайте активировано несколько языков и пользователь выбрал язык в форме настройки учетной записи
picture	Путь к файлу изображения, которое пользователь назначил для своей учетной записи
init	Почтовый адрес, введенный пользователем при первоначальной регистрации
data	Произвольные данные, которые могут храниться в этом поле модулями
Из таблицы <i>user_roles</i>	
roles	Роли, назначенные в данный момент сеансу пользователя системой PHP
Из таблицы <i>sessions</i>	
sid	Идентификатор, назначенный сеансу данного пользователя системой PHP
ssid	Идентификатор, назначенный защищенному сеансу пользователя системой PHP
hostname	IP-адрес, с которого пользователь просматривает текущую страницу
timestamp	Метка времени Unix для момента, когда браузер пользователя в последний раз получил заполненную страницу
cache	Метка времени, используемая для кэширования отдельных пользователей (см. <code>includes/cache.inc</code>)
session	Произвольные временные данные, которые могут сохраняться здесь модулями во время сеанса пользователя

Проверка, выполнил ли пользователь вход

Стандартный способ узнать, выполнил ли пользователь вход — проверка, равно ли нулю значение `$user->uid`. Для этой цели в Drupal предусмотрена удобная функция `user_is_logged_in()` (и аналогичная функция `user_is_anonymous()`):

```
if (user_is_logged_in()) {
    $output = t('User is logged in.');
```

// Пользователь выполнил вход

```
else {
    $output = t('User is an anonymous user.');
```

// Пользователь является анонимным

```
}
```

Хуки пользователя

Реализация хуков пользователя позволяет модулям реагировать на различные действия, выполняемые с пользовательской учетной записью, и изменять объект `$user`. Имеются несколько вариантов функции `hook_user`, и каждый вариант предназначен для выполнения своего действия (табл. 6.2).

Таблица 6.2. Функции `hook_user`

Функция хука	Назначение
<code>hook_username_alter(&\$name, \$account)</code>	Изменение пользовательского имени, отображаемого для пользователя
<code>hook_user_cancel(\$edit, \$account, \$method)</code>	Вызывается при отменах пользовательских учетных записей
<code>hook_user_cancel_methods_alter(&\$methods)</code>	Изменение метода отмены учетной записи
<code>hook_user_categories()</code>	Получение списка параметров пользователя или изменений в информации профиля
<code>hook_user_delete(\$account)</code>	Реакция на удаление пользователя
<code>hook_user_insert(&\$edit, \$account, \$category)</code>	После создания пользовательской учетной записи
<code>hook_user_load(\$users)</code>	Работает с объектами пользователей при загрузке из базы данных
<code>hook_user_login(&\$edit, \$account)</code>	Сразу после входа пользователя
<code>hook_user_logout(\$account)</code>	Сразу после выхода пользователя
<code>hook_user_operations()</code>	Массовые операции с пользователями
<code>hook_user_presave(&\$edit, \$account, \$category)</code>	Непосредственно перед созданием или изменением пользовательской учетной записи
<code>hook_user_role_delete(\$role)</code>	Информирование других модулей об удалении пользовательской роли
<code>hook_user_role_insert(\$role)</code>	Информирование других модулей о добавлении пользовательской роли
<code>hook_user_role_update(\$role)</code>	Информирование других модулей об изменении пользовательской роли
<code>hook_user_update(&\$edit, \$account, \$category)</code>	После изменения пользовательской учетной записи
<code>hook_user_view(\$account, \$viewmode)</code>	При отображении информации пользовательской учетной записи
<code>hook_user_view_alter(&\$build)</code>	После создания пользователя; модуль может изменить структурированное содержимое

Внимание! Не путайте параметр `$account` во многих функциях хуков пользователя с глобальным объектом `$user`. Параметр `$account` представляет собой объект `$user` для обрабатываемой в данный момент учетной записи. Глобальный объект `$user` — это пользователь, который в данный момент выполнил вход. Обычно они совпадают, но не всегда.

Функция `hook_user_view($account, $view_mode)`

Функция `hook_user_view($account, $view_mode)` используется модулями для добавления информации в страницы профиля пользователя — например, то, что отображается на странице `http://example.com/?q=user/1`, показанной на рис. 6.1.

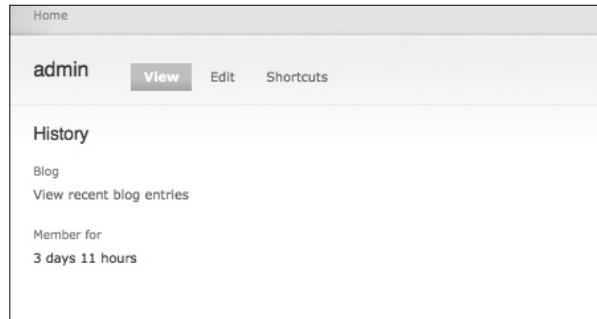


Рис. 6.1. Страница профиля пользователя с модулем блога и модулем пользователя, в котором реализована `hook_user_view()` для добавления дополнительной информации

Рассмотрим, как модуль блога добавил свою информацию на эту страницу с помощью функции `hook_user_view()`:

```
/**
 * Реализация hook_user_view().
 */
function blog_user_view($account) {
  if (user_access('create blog content', $account)) {
    $account->content['summary']['blog'] = array(
      '#type' => 'user_profile_item',
      '#title' => t('Blog'),
      '#markup' => l(t('View recent blog entries'),
        // Просмотреть последние записи блога
        "blog/$account->uid", array('attributes'
          => array('title' => t("Read !username's latest blog entries.",
            // Прочитать последние записи блога пользователя !username
            array('!username' => format_username($account))))),
      '#attributes' => array('class' => array('blog')),
    );
  }
}
```

Функция `view` заносит некоторую информацию в компонент `$user->content`. Информация пользовательского профиля разбита на категории, и каждая категория представляет собой страницу информации о пользователе. На рис. 6.1 показана лишь одна категория `History` (История). Внешний массив должен быть индексирован именем категории. В данном примере использован ключ `summary`, который соответствует категории `History` (конечно, лучше, если имена ключа и категории совпадают). Внутренний массив (или массивы) должен содержать уникальный текстовый ключ (в данном случае

blog) и элементы #type, #title, #markup и #attributes. Тип user_profile_item указывает уровню темизации Drupal на modules/user/user-profile-item.tpl.php. Сравнив приведенный фрагмент кода с рис. 6.1, можно понять, как отображаются эти элементы. В листинге 6.1 показано содержимое массива \$user->content, который и отображается на рис. 6.1.

Листинг 6.1. Структура \$user->content

```
Array
(
    [#pre_render] => Array
        (
            [0] => _field_extra_fields_pre_render
        )
    [#entity_type] => user
    [#bundle] => user
    [#attached] => Array
        (
            [css] => Array
                (
                    [0] => modules/field/theme/field.css
                )
        )
    [summary] => Array
        (
            [blog] => Array
                (
                    [#type] => user_profile_item
                    [#title] => Blog // Блог
                    [#markup] => View recent blog entries // Посмотреть последние
                                                                // записи блога
                    [#attributes] => Array
                        (
                            [class] => Array
                                (
                                    [0] => blog
                                )
                        )
                )
            [#type] => user_profile_category
            [#attributes] => Array
                (
                    [class] => Array
                        (
                            [0] => user-member
                        )
                )
            [#weight] => 5
            [#title] => History // История
            [member_for] => Array
                (
                    [#type] => user_profile_item
                    [#title] => Member for // Член
                    [#markup] => 3 days 11 hours // 3 дня 11 часов
                )
        )
)
```

```
[user_picture] => Array
(
    [#markup] =>
    [#weight] => -10
)
```

В модуле может быть также реализована функция `hook_user_view()` для обработки элементов профилей в массиве `$user->content` перед применением темы. Ниже приведен пример, в котором элемент профиля блога просто удален со страницы профиля пользователя. Имя выбрано для функции, которая находится в гипотетическом модуле `hide.module`:

```
/**
 * Реализация hook_user_view().
 */
function hide_user_view($account, $view_mode = 'full') {
    unset($account->content['summary']['blog']);
}
```

Процесс регистрации пользователя

По умолчанию для регистрации пользователя на сайте Drupal нужно лишь ввести пользовательское имя и допустимый адрес электронной почты. Модули могут добавлять в форму регистрации пользователя собственные поля, реализуя несколько хуков пользователя. Сейчас мы напишем модуль с именем `legalagree.module`, который поможет сайту выжить в полном опасностей современном мире.

Сначала создайте папку `sites/all/modules/custom/legalagree` и добавьте в каталог `legalagree` файлы, описанные в листингах 6.2 и 6.3. Затем активируйте модуль с помощью пункта меню `Administer` ⇒ `Site building` ⇒ `Modules` (Администрирование ⇒ Структура сайта ⇒ Модули).

Листинг 6.2. Файл `legalagree.info`

```
name = Legal Agreement
description = Displays a dubious legal agreement during user registration.
package = Pro Drupal Development
core = 7.x
files[] = legalagree.module
```

Листинг 6.3. Файл `legalagree.module`

```
<?php
/**
 * @file
 * Поддержка коварного текста официального соглашения
 * во время регистрации пользователя.
 */

/**
 * Реализация hook_form_alter().
 */
function legalagree_form_alter(&$form, &$form_state, $form_id) {
```

```

// Проверка, выполняет ли форма регистрацию или настройку профиля пользователя.
// Если нет – возврат без каких-либо действий.
if (!($form_id == 'user_register_form' || $form_id == 'user_profile_form'))
{
    return;
}

// Добавление в форму новой функции проверки для заключения официального соглашения.
$form['#validate'][] = 'legalagree_user_form_validate';

// Добавление набора полей для заключения официального соглашения.
$form['account']['legal_agreement'] = array(
    '#type' => 'fieldset',
    '#title' => t('Legal agreement') Официальное соглашение
);

// Добавление кнопок переключателя для заключения соглашения.
$form['account']['legal_agreement']['decision'] = array(
    '#type' => 'radios',
    '#description' => t('By registering at %site-name, you agree that
at any time, we (or our surly, brutish henchmen) may enter your place of
residence and smash your belongings with a ball-peen hammer.'),

// Регистрируясь на сайте %site-name, вы соглашаетесь с тем, что в любое
// время мы (или наши угрюмые и свирепые бандиты) можем прийти к вам и
// разломать все ваши вещи кувалдой.
array('%site-name' => variable_get('site_name', 'drupal'))),
    '#default_value' => 0,
    '#options' => array(t('I disagree'), t('I agree'))
);

}

/**
 * Обработчик проверки формы для текущего пароля на user_account_form().
 *
 * @see user_account_form()
 */
function legalagree_user_form_validate($form, &$form_state) {

    global $user;

    // Визуальная проверка.
    if ($form_state['input']['decision'] <> 1) {
        form_set_error('decision', t('You must agree to the Legal Agreement before
registration can be completed.'));
    }

    // Для завершения регистрации вы должны принять официальное соглашение.
    } else {
        watchdog('user', t('User %user agreed to legal terms',
            array('%user' => $user->name)));
        // Пользователь %user принял официальные условия
    }
}

```

Хук пользователя вызывается во время создания регистрационной формы, во время проверки данных этой формы и после добавления записи о пользователе в базу данных. Наш короткий модуль создаст примерно такую регистрационную форму, как показанная на рис. 6.2.

Рис. 6.2. Измененная форма регистрации пользователя

Сбор информации о пользователях с помощью `profile.module`

Если вы планируете добавить в форму регистрации пользователя сбор информации о пользователях, то стоит сначала попробовать модуль `profile.module`, а уже потом решать, нужно ли писать собственный модуль. Этот модуль позволяет создавать произвольные формы для сбора данных, определять, какая информация требуется и/или предназначена для сбора в форме регистрации пользователя, а также указать степень конфиденциальности этой информации. Кроме того, он позволяет администратору определять страницы для просмотра пользователей по вариантам их выбора с помощью URL-адреса, составленного из адреса сайта с добавлением `profile/` плюс имя поля профиля/ плюс значение.

Например, если определить текстовое поле профиля с именем `profile_color`, то потом можно просмотреть всех пользователей, которым понравился синий цвет, на странице http://example.com/?q=profile/profile_color/black. Или, допустим, вы создаете веб-сайт для конференции и отвечаете за заказ ужина для участников. Можно определить поле-флажок с именем `profile_vegetarian` и потом просмотреть всех пользователей-вегетарианцев по адресу http://example.com/?q=profile/profile_vegetarian (обратите внимание, что для полей-флажков значение задается неявно и поэтому игнорируется, т.е. к URL не добавляется никакое значение вроде `black` из поля `profile_color`).

А вот вполне реальный пример: список пользователей на сайте <http://drupal.org>, которые посетили конференцию по Drupal в Сан-Франциско в 2010 г., можно просмотреть в каталоге `profile/conference-sf-2010` (в данном случае перед именем поля нет префикса `profile_`).

Совет. Автоматическое создание страниц итогов по профилям работает, только если заголовок поля Page (Страница) заполнен в параметрах поля профиля и не доступно для полей с текстом, URL-адресом или датой.

Процесс входной регистрации

Процесс входной регистрации начинается тогда, когда пользователь заполняет входную форму (обычно по адресу `http://example.com/?q=user` или в блоке) и щелкает на кнопке Log in (Вход).

Подпрограммы проверки входной формы проверяют, не заблокировано ли имя пользователя, не запрещен ли доступ правилом доступа и ввел ли пользователь корректные имя и пароль. Об обнаружении любого из этих противопоказаний выдается сообщение.

На заметку! В Drupal применяется как локальная, так и внешняя аутентификация. Примерами систем внешней аутентификации могут служить OpenID, LDAP, Pubcookie и др.

Drupal пытается локально выполнить подключение пользователя, для чего ищет строку в таблице `users` с совпадающими именем и хешем пароля. При успешно выполненном входе запускаются два хука пользователя (`load` и `login`), которые могут быть реализованы в ваших модулях, как показано на рис. 6.3.

Добавление данных в объект `$user` во время загрузки

Операция `load` хука пользователя вызывается, когда объект `$user` успешно загружен из базы данных в результате вызова `user_load()`. Это происходит, когда пользователь выполнил вход, когда из ноды выбрана информация об авторе и в ряде других ситуаций.

На заметку! В силу трудоемкости вызова хука пользователя, функция `user_load()` не вызывается, когда создается текущий экземпляр объекта `$user` для запроса (см. раздел “Объект `$user`” ранее в главе). Если вы пишете собственный модуль, всегда вызывайте `user_load()` перед вызовом функции, которая требует наличия полностью загруженного объекта `$user`, если вы не полностью уверены, что он загружен.

Сейчас мы напишем модуль с именем `loginhistory`, который будет вести хронологию входов пользователя. Он будет выводить количество входов пользователя на странице My account (Моя учетная запись). Создайте папку `loginhistory` в каталоге `sites/all/modules/custom/` и добавьте туда файлы, приведенные в листингах 6.4–6.6. Первым будет файл `sites/all/modules/custom/loginhistory/loginhistory.info`.

Листинг 6.4. Файл `loginhistory.info`

```
name = Login History
description = Keeps track of user logins.
package = Pro Drupal Development
core = 7.x
files[] = loginhistory.install
files[] = loginhistory.module
```

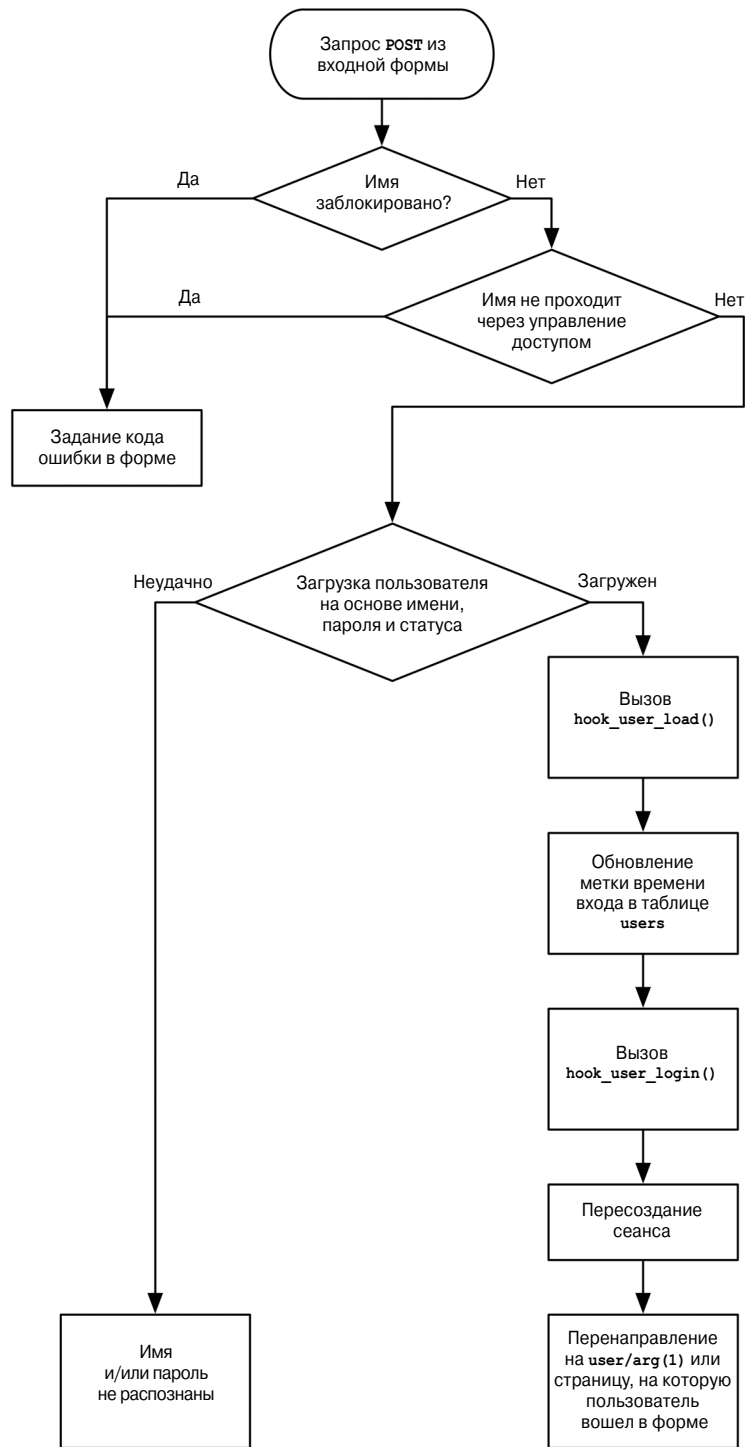


Рис. 6.3. Выполнение входной регистрации пользователя

Нам понадобится файл `.install` для создания таблицы в базе данных, чтобы хранить информацию о входах — `sites/all/modules/custom/loginhistory/loginhistory.install`.

Листинг 6.5. Файл `loginhistory.install`

```
<?php

/**
 * Реализация hook_schema().
 */
function loginhistory_schema() {
  $schema['login_history'] = array(
    'description' => 'Stores information about user logins.',
    // Хранит информацию о входах пользователя
    'fields' => array(
      'uid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => 'The {user}.uid of the user logging in.',
        // {user}.uid пользователя, выполняющего входную регистрацию.
      ),
      'login' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => 'Unix timestamp denoting time of login.',
        // Метка времени Unix о времени входа.
      ),
    ),
    'index' => array('uid'),
  );
  return $schema;
}
```

Листинг 6.6. Файл `loginhistory.module`

```
<?php
/**
 * @file
 * Отслеживает входы пользователя.
 */

/**
 * Реализация hook_user_login.
 */
function loginhistory_user_login(&$edit, $account) {

  // Добавление новой записи при каждом входе пользователя.
  $nid = db_insert('login_history')->fields(array(
    'uid' => $account->uid,
    'login' => $account->login
  ))->execute();
}
```

```

/**
 * Реализация hook_user_view_alter.
 */
function loginhistory_user_view_alter(&$build){

  global $user;

  // Подсчет входов пользователя.
  $login_count = db_query("SELECT count(*) FROM {login_history} where uid = :uid",
    array(':uid' => $user->uid))->fetchField();

  // Добавление на страницу пользователя количества его входов
  $build['summary']['login_history'] = array(
    '#type' => 'user_profile_item',
    '#title' => t('Number of logins'), // Количество входов
    '#markup' => $login_count,
    '#weight' => 10,
  );
}

```

После инсталляции этого модуля каждая успешная входная регистрация пользователя будет запускать операцию login хука `hook_user_login`, на которую модуль будет отвечать вставкой записи в таблицу `login_history` базы данных. Когда во время работы хука `hook_user_view` загружается объект `$user`, будет вызвана функция `hook_user_view_alter`, и модуль добавит текущее количество входов для данного пользователя на просматриваемую страницу *My account* (Моя учетная запись), как показано на рис. 6.4.

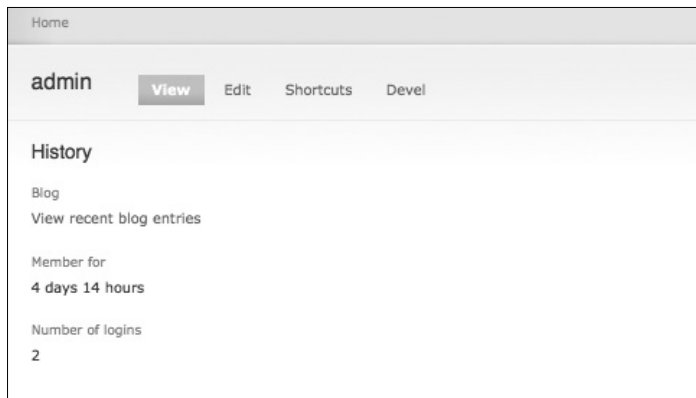


Рис. 6.4. История входов, где зафиксированы входные регистрации пользователя

Предоставление категорий пользовательской информации

Если у вас есть учетная запись на сайте <http://drupal.org>, вы можете увидеть эффект предоставления категорий пользовательской информации. Для этого выполните вход и щелкните на ссылке *My account* (Моя учетная запись); на открывшейся странице вы можете предоставить информацию о себе в нескольких категориях: участие в разработке Drupal, личная информация, рабочая информации и желание получать извещения о новостях.

Внешний вход

Иногда использовать локальную таблицу Drupal users не требуется. Например, возможно, таблица с данными о пользователях уже имеется в другой базе данных или в LDAP. Среда Drupal позволяет легко интегрировать внешнюю аутентификацию в процесс входной регистрации.

Сейчас мы реализуем очень простой модуль внешней аутентификации, чтобы продемонстрировать работу внешней аутентификации. Предположим, что в вашей компании принимаются на работу только люди по имени Дэйв (Dave), а пользовательские имена назначаются на основе имени и фамилии. Этот модуль аутентифицирует любого с пользовательским именем, начинающимся на dave, т.е. пользователи davebrown, davesmith и davejones успешно выполняют вход. Мы воспользуемся функцией `form_alter()` для изменения обработчика проверки входа пользователя, чтобы выполнять собственный обработчик проверки. Вот содержимое файла `sites/all/modules/custom/authdave/authdave.info`:

```
name = Authenticate Daves
description = External authentication for all Daves.
package = Pro Drupal Development
core = 7.x
files[] = authdave.module
```

А вот сам модуль `authdave.module`:

```
<?php

/**
 * Реализация hook_form_alter().
 * Заменяем локальный обработчик входной проверки нашим обработчиком.
 */
function authdave_form_alter(&$form, &$form_state, $form_id) {
  // В данном простом примере выполняется аутентификация по имени
  // пользователя — проверяем, начинается ли оно на dave.
  if ($form_id == 'user_login' || $form_id == 'user_login_block') {
    $form['#validate'][] = 'authdave_user_form_validate';
  }
}

/**
 * Специальная функция проверки формы.
 */
function authdave_user_form_validate($form, &$form_state) {
  if (!authdave_authenticate($form_state)) {
    form_set_error('name', t('Unrecognized username.'));
    // Пользовательское имя не распознано.
  }
}

/**
 * Специальная функция аутентификации пользователя.
 */
function authdave_authenticate($form_state) {
  // Извлечение первых четырех символов пользовательского имени.

  $username = $form_state['input']['name'];
  $testname = drupal_substr(drupal_strtolower($username), 0, 4);
```

```
// Проверка, зовут ли пользователя dave.
if ($testname == "dave") {
  // Если это Dave, вызывается функция external_login_register —
  // либо для входа, либо для создания новой учетной записи,
  // если тот, кто вошел, еще не является пользователем Drupal.
  user_external_login_register($username, 'authdave');
  return TRUE;
} else {
  return FALSE;
}
}
```

В модуле authdave (рис. 6.5) мы просто заменяем второй обработчик проверки собственным обработчиком. Сравните рис. 6.5 с рис. 6.3, где показан процесс локальной входной регистрации пользователя.

Функция `user_external_login_register()` — вспомогательная функция, которая регистрирует пользователя, если это его первый вход, а затем выполняет его вход. Схема выполнения приведена на рис. 6.6 для гипотетического пользователя davejones, который вошел впервые.

Если имя пользователя начинается на dave, и это первый вход пользователя, т.е. в таблице `users` нет строки для этого пользователя, такая строка создается. Однако при этом не заносится адрес электронной почты, как при стандартной локальной регистрации пользователя, поэтому такой простой модуль не является приемлемым решением, если для работы сайта необходима рассылка почтовых сообщений пользователям. Тогда все-таки придется заполнить столбец `mail` в таблице `users`, а для этого нужно, чтобы модуль реагировал на операцию `insert` в хуке пользователя, который выполняется при вставке нового пользователя:

```
/**
 * Реализация hook_user().
 */
function authdave_user_insert(&$edit, &$account, $category = NULL) {
  global $authdave_authenticated;
  if ($authdave_authenticated) {
    $email = mycompany_email_lookup($account->name);

    // Занесение адреса электронной почты для этого
    // пользователя в таблицу users.
    db_update('users')
      ->fields(
        array(
          'mail' => $email,
        )
      )
      ->condition('uid', $account->uid)
      ->execute();
  }
}
```

Внимательные читатели заметят, что код не может распознать, когда выполняется локальная, а когда внешняя аутентификация, поэтому мы предусмотрительно завели глобальную переменную, означающую, что наш модуль выполнил аутентификацию. Кроме того, можно выполнить запрос к таблице `authmap`:

```
db_query("SELECT uid FROM {authmap} WHERE uid = :uid AND module = :module",
  array(':uid' => $account->uid, 'module' => 'authdave');
```

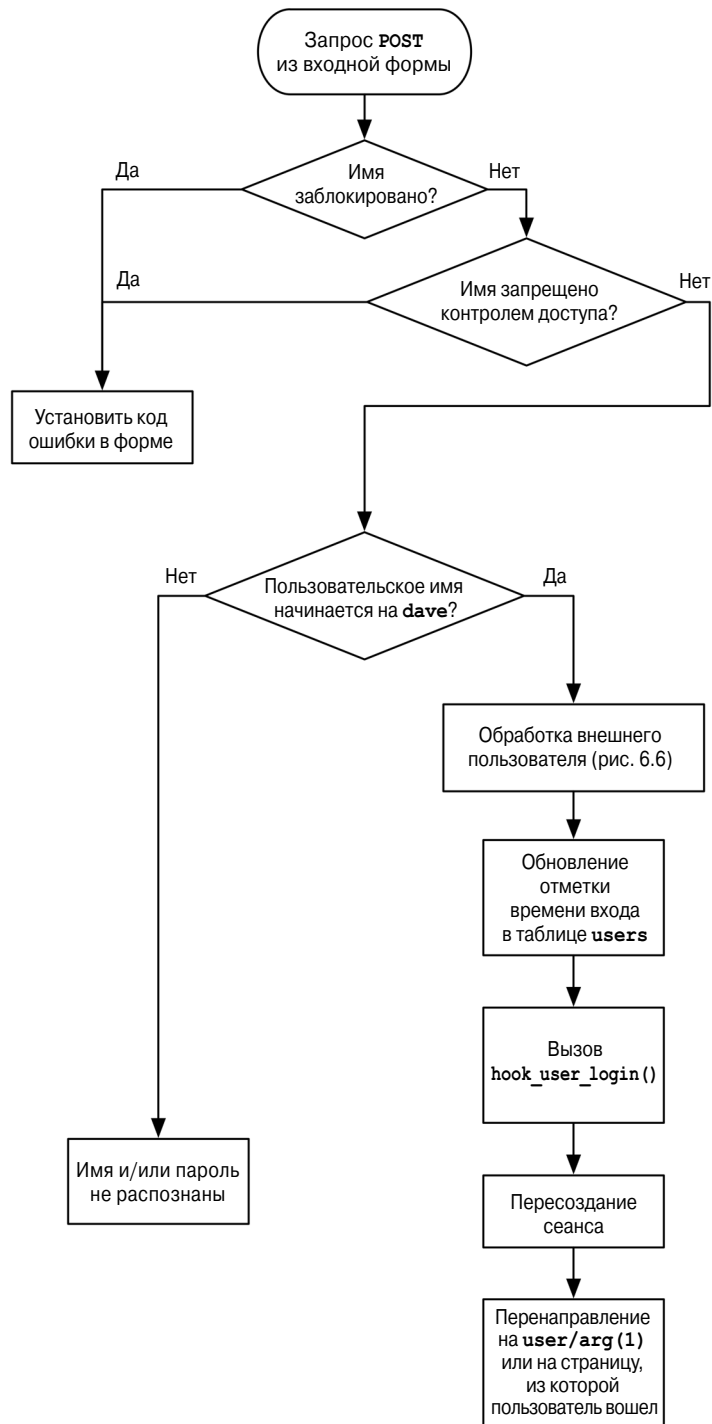


Рис. 6.5. Выполнение внешнего входа, где второй обработчик проверки предоставлен модулем `authdave` (сравните с рис. 6.3)

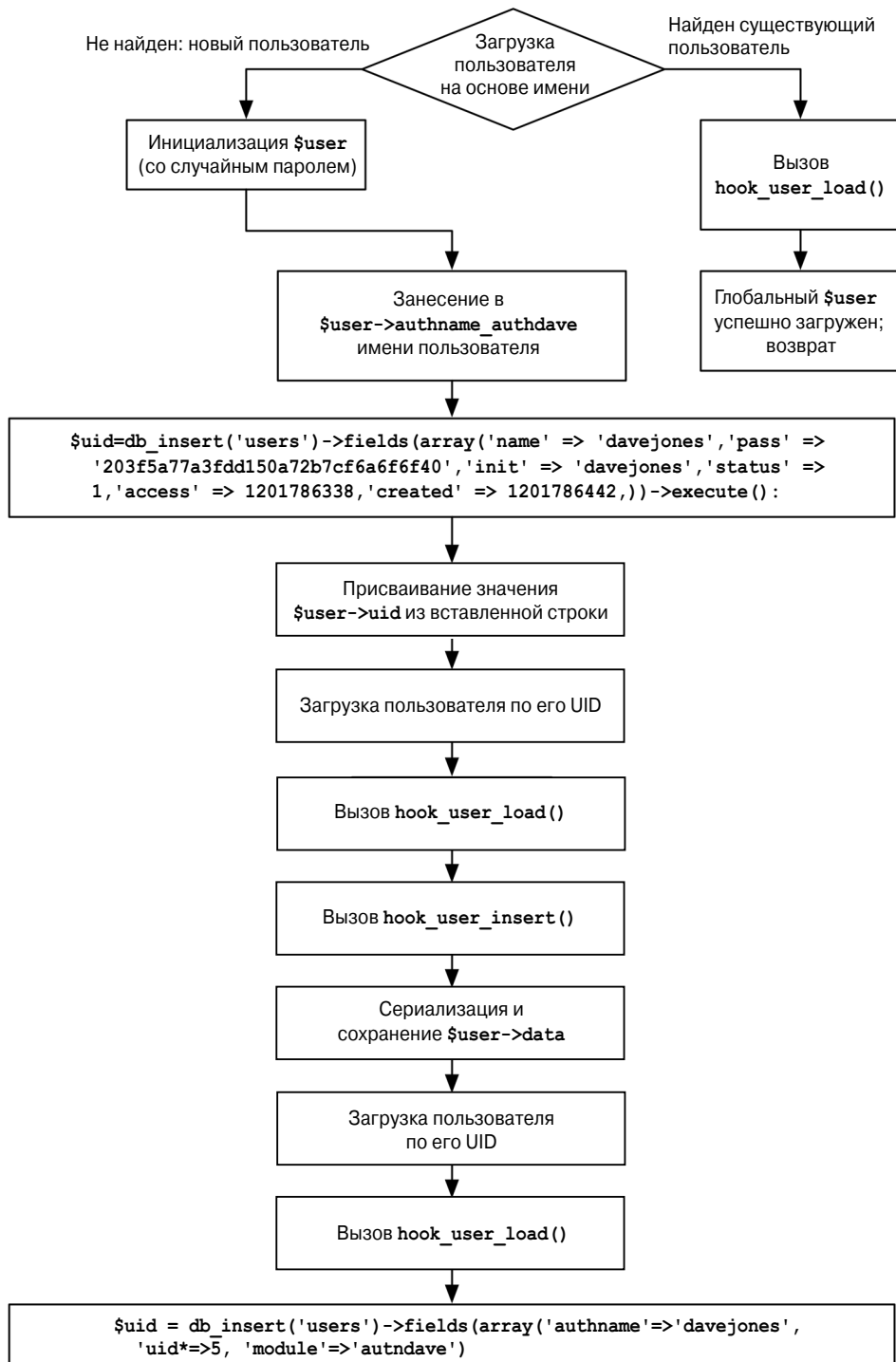


Рис. 6.6. Схема процесса внешнего входа/регистрации пользователя

Для всех пользователей, добавленных с помощью внешней аутентификации, будет присутствовать строка как в таблице `authmap`, так и в таблице `users`. Однако в данном случае аутентификация и `hook_user_insert` выполняются во время одного и того же запроса, так что глобальная переменная является хорошей альтернативой запросу к базе данных.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Понимать, как пользователи представляются внутри Drupal.
- Знать, как разными способами сохранить информацию, связанную с пользователем.
- Уметь вмешаться в процесс регистрации пользователя, чтобы получить больше информации от зарегистрированного пользователя.
- Уметь вмешаться в процесс входа пользователя, чтобы выполнить во время входа собственный код.
- Понимать, как работает внешняя аутентификация пользователей.
- Уметь реализовать собственный модуль внешней аутентификации.

Дополнительную информацию о внешней аутентификации можно почерпнуть в модулях `openid.module` (часть ядра Drupal) и `pubcookie.module` (дополнительный модуль).

ГЛАВА 7

Работа с нодами

В этой главе вы ознакомитесь с нодами и типами нод. Вы научитесь создавать тип ноды двумя различными способами. Вначале мы рассмотрим программное решение — напомним модуль, использующий хуки Drupal. Этот подход обеспечивает больший контроль и гибкость при определении, что нода может делать, а чего не может. Затем мы увидим, как создавать тип ноды из интерфейса администрирования Drupal. И в конце главы мы рассмотрим механизм управления доступом к нодам Drupal.

Совет. Разработчики часто используют термины *нода* (node) и *тип ноды* (node type). В пользовательском интерфейсе Drupal они называются соответственно *сообщениями* (post) и *типами контента* (content type), чтобы говорить на одном языке с администраторами сайтов.

Так что же собой представляет нода?

Один из первых вопросов, которые задают новички в разработке на Drupal — что такое нода? Нода представляет собой фрагмент контента. Drupal присваивает каждому такому фрагменту контента идентификационный номер, который называется *идентификатором ноды* (node ID, а в коде — `$nid`). Обычно у нод имеется название (заголовок), которое позволяет администратору просмотреть список нод по их названиям.

На заметку! Если вы знакомы с объектно-ориентированным программированием, можете считать тип ноды классом, а отдельные ноды — экземплярами класса. Однако код Drupal не полностью объектно-ориентирован, и на то имеются веские причины (см. <http://api.drupal.org/api/HEAD/file/developer/topics/oop.html>).

Существует множество различных разновидностей нод, или *типов нод*. Обычно используемыми типами нод являются “запись блога”, “опрос” и “форум”. Часто в качестве синонима для *типа ноды* применяется термин *тип контента* (content type), хотя тип ноды является более абстрактной концепцией, и его можно считать порожденным от базовой ноды, как показано на рис. 7.1.

Тот факт, что все типы контента являются нодами, удобен тем, что они основаны на одной и той же базовой структуре данных. Для разработчиков это означает, что для многих операций весь контент можно считать однотипным с программистской точки зрения. С нодами легко выполнять пакетные операции, и для собственноручно разработанных типов контента сразу же доступно много готовых функций. Поиск, создание, редактирование и управление контентом — эти действия поддерживаются самой средой Drupal на основе базовой структуры данных нод и ее поведения. Это единообразие видно и конечным пользователям. Формы для создания, редактирования и удаления нод выглядят и ведут себя одинаково, что приводит к однотипным и поэтому легко осваиваемым интерфейсам.

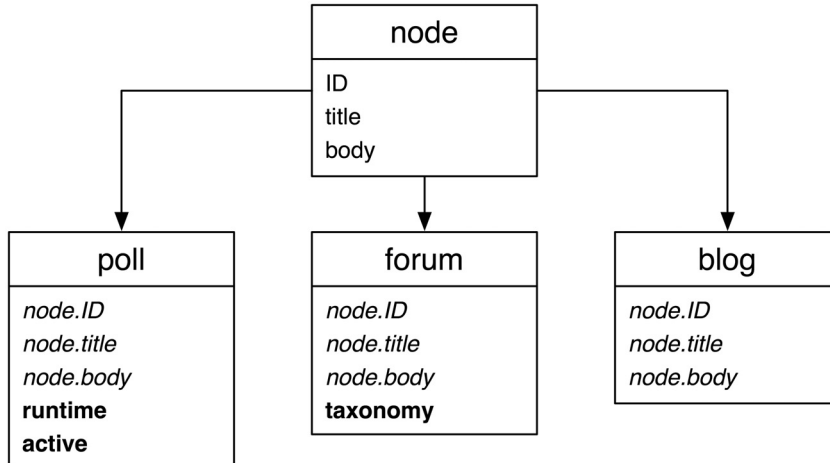


Рис. 7.1. Типы нод порождаются от базовой ноды и могут иметь дополнительные поля

Типы нод расширяют базовую ноду, обычно за счет добавления собственных атрибутов данных. Нода типа *poll* (опрос) хранит параметры голосования — продолжительность опроса, активен ли опрос в данный момент и могут ли голосовать отдельные пользователи. Нода типа *forum* загружает для каждой ноды таксонометрический термин, чтобы знать, где она находится среди всех форумов, определенных администратором. А ноды типа *blog* (блог) не добавляют никаких дополнительных данных. Вместо этого они просто добавляют различные представления данных, создавая блоги для каждого пользователя и агрегаторы RSS для каждого блога. У всех нод имеются следующие атрибуты, хранимые в таблицах *node* и *node_revisions* базы данных.

- *nid*. Уникальный идентификатор ноды.
- *vid*. Уникальный идентификатор версии для ноды. Он необходим потому, что Drupal может хранить ревизии контента для нод. Значения *vid* уникальны для всех нод и ревизий нод.
- *type*. Каждая нода относится к какому-то типу ноды, например, *blog*, *story*, *article*, *image* и т.д.
- *language*. Язык ноды. Первоначально этот столбец пуст, что обозначает нейтральную к языку ноду.
- *title*. Короткая (не более 255 символов) строка, которая используется как заголовок ноды. В типе ноды можно также объявить, что она не имеет заголовка, для чего в поле *has_title* таблицы *node_type* необходимо указать значение 0.
- *uid*. Идентификатор пользователя-автора ноды. По умолчанию нода имеет одного автора.
- *status*. Значение 0 означает, что нода не опубликована, т.е. контент скрыт от тех, у кого нет права «администрирование нод». Значение 1 означает, что нода опубликована, и ее контент видим для пользователей с правами «доступ к контенту». Отображение опубликованной ноды может быть заблокировано системой управления доступом Drupal уровня нод (см. разделы «Ограничение доступа к типу ноды с помощью *hook_access()*» и «Ограничение доступа к нодам» далее в главе). Опубликованная нода индексируется модулем поиска, если он активирован.
- *created*. Метка времени Unix о моменте создания ноды.

- **changed.** Метка времени Unix о моменте последнего изменения ноды. При использовании системы ревизий нод это же значение находится и в поле `timestamp` таблицы `node_revisions`.
- **comment.** Целочисленное поле, которое описывает состояние комментариев этой ноды. Возможны три различных значения.
 - 0. Для ноды комментарии отключены. Это стандартное значение для существующих нод, когда отключен модуль комментирования. В разделе **Comment settings** (Параметры комментирования) в пользовательском интерфейсе формы редактирования это указывается значением **Disabled** (Отключено).
 - 1. Для ноды запрещено добавлять комментарии. В разделе **Comment settings** (Параметры комментирования) в пользовательском интерфейсе формы редактирования это указывается значением **Read only** (Только чтение).
 - 2. Комментарии можно просматривать и добавлять. Управление тем, кто может помещать комментарии и как комментарии выглядят, возлагается на модуль комментирования. В разделе **Comment settings** (Параметры комментирования) в пользовательском интерфейсе формы редактирования это указывается значением **Read/Write** (Чтение/запись).
- **promote.** Целочисленное поле, определяющее, нужно ли выводить ноду на передней странице. Возможны два значения.
 - 1. Перемещена на переднюю страницу. Нода перемещена на стандартную переднюю страницу сайта. Нода по-прежнему будет выглядеть как обычная страница — например, `http://example.com/?q=node/3`. Следует отметить, что поскольку на странице **Configuration** ⇒ **Site information** (Настройка ⇒ Информация о сайте) можно изменять переднюю страницу сайта, сам термин “передняя страница” может ввести в заблуждение. Точнее будет сказать, что страница `http://example.com/?q=node` будет содержать все ноды со значением 1 в поле `promote`. По умолчанию URL-адрес `http://example.com/?q=node` считается передней страницей.
 - 0. Нода не выводится на странице `http://example.com/?q=node`.
- **sticky.** Когда Drupal выводит список нод на странице, по умолчанию сначала выводятся ноды, помеченные как **sticky** (“приклеенные”), а затем остальные “не приклеенные” ноды в порядке времени создания. Другими словами, ноды **sticky** “приклеены” к началу списка нод. Значение 1 означает **sticky**, а 0 — обычную ноду. В одном списке может быть несколько нод **sticky**.
- **tnid.** Если нода является переведенной версией другой ноды, то здесь хранится `nid` ноды на исходном языке. Например, если нода 3 написана на английском языке, а нода 5 содержит тот же контент, но на русском, то поле `tnid` ноды 5 содержит значение 3.
- **translate.** Значение 1 означает, что перевод необходимо обновить, а 0 — что перевод актуален.

При использовании системы ревизий нод Drupal создает ревизии контента, а также отслеживает, кто выполнил последнее изменение.

Не все является нодами

Пользователи, блоки и комментарии не являются нодами. Каждая из этих специализированных структур данных имеет собственную систему хуков в соответствии с ее назначением. Ноды (обычно) имеют заголовок и тело контента, а в структуре данных,

представляющей пользователя, это не требуется. Зато пользователям нужен адрес электронной почты, пользовательское имя и надежный способ хранения паролей. Блоки — это облегченное решение для хранения небольших объемов контента, таких как меню навигации, поле поиска, список последних комментариев и т.д. Комментарии тоже не являются нодами, и поэтому они также относятся к облегченной категории. Вполне возможно, что на нескольких страницах может находиться по 100 и более комментариев, и если каждый из этих комментариев будет проходить при загрузке через систему хуков ноды, это просто катастрофически снизит производительность.

В прошлом было немало споров, должны ли быть пользователи или комментарии нодами, и в некоторых разработанных модулях они действительно представлены как ноды. Учтите, что возврат к этому спору аналогичен крикам “Emsac лучше!” в ответ на требование следовать программистскому соглашению.

Создание модуля ноды

Если вам понадобится создать в Drupal новый тип контента, то, следуя традиции, придется написать *модуль ноды*, который будет отвечать за выполнение и предоставление новых и интересных вещей, необходимых этому типу контента. Здесь сказано “следуя традиции”, т.к. последние веяния в среде Drupal позволяют создавать типы контента в интерфейсе администрирования, расширяя их возможности добровольно разработанными модулями — без написания модуля с самого начала. В настоящей главе рассматриваются оба этих решения.

Мы напишем модуль ноды, который позволит пользователям добавлять на сайт объявления о рабочих вакансиях. Нода вакансии будет содержать заголовок, тело с текстом объявления и поле для ввода названия компании. Для заголовка и тела объявления будут использованы встроенные ноды `title` и `body`, стандартные для всех нод Drupal. А для названия компании потребуется новое специальное поле.

Начнем с создания папки с именем `job_post` в каталоге `your sites/all/modules/custom`.

Создание файла `.install`

Файл `.install` для модуля вакансий выполняет все операции по установке: определение типа ноды, создание полей, составляющих новый тип ноды, и выполнение процесса деинсталляции модуля:

```
<?php
/**
 * @file
 * Инсталляция файла для модуля Job Post.
 */

/**
 * Реализация hook_install().
 * - Добавление поля тела.
 * - Настройка поля тела.
 * - Создание поля для названия компании.
 */
function job_post_install() {
  node_types_rebuild();
  $types = node_type_get_types();
  // Добавление поля тела в тип ноды.
  node_add_body_field($types['job_post']);
  // Загрузка определения экземпляра для тела типа контента.
  $body_instance = field_info_instance('node', 'body', 'job_post');
```

```

// Настройка поля тела.
$body_instance['type'] = 'text_summary_or_trimmed';

// Сохранение изменений в экземпляре поля тела.
field_update_instance($body_instance);

// Создание всех полей, добавляемых в тип контента.
foreach (_job_post_installed_fields() as $field) {
  field_create_field($field);
}

// Создание всех экземпляров полей.
foreach (_job_post_installed_instances() as $instance) {
  $instance['entity_type'] = 'node';
  $instance['bundle'] = 'job_post';
  field_create_instance($instance);
}
}

/**
 * Возврат структурированного массива с определением полей,
 * созданного данным типом контента.
 * В модуле вакансий имеется лишь одно дополнительное поле - название компании.
 * Другие поля можно добавить, определив их в данной функции как
 * дополнительные элементы в приведенном ниже массиве.
 */
function _job_post_installed_fields() {
  $t = get_t();
  return array(
    'job_post_company' => array(
      'field_name' => 'job_post_company',
      'label'      => $t('Company posting the job listing'),
                  // Компания, поместившая объявление
      'type'       => 'text',
    ),
  );
}

/**
 * Возврат структурированного массива с определениями экземпляров полей,
 * связанных с данным типом контента.
 */
function _job_post_installed_instances() {
  $t = get_t();
  return array(
    'job_post_company' => array(
      'field_name' => 'job_post_company',
      'type'       => 'text',
      'label'      => $t('Company posting the job listing'),
      'widget'     => array(
        'type'     => 'text_textfield',
      ),
      'display'    => array(
        'example_node_list' => array(
          'label' => $t('Company posting the job listing'),

```

```

        'type'      => 'text',
    ),
),
);
}
/**
 * Реализация hook_uninstall().
 */
function job_post_uninstall() {
    // Сбор всего контента, который может быть создан при активации этого модуля.
    $sql = 'SELECT nid FROM {node} n WHERE n.type = :type';
    $result = db_query($sql, array(':type' => 'job_post'));
    $nids = array();
    foreach ($result as $row) {
        $nids[] = $row->nid;
    }

    // Одновременное удаление всех нод.
    node_delete_multiple($nids);
    // Перебор всех определенных в модуле полей и удаление
    // всех экземпляров полей, их данных и самих полей.
    foreach (array_keys(_job_post_installed_fields()) as $field) {
        field_delete_field($field);
    }

    // Проход в цикле по всем остальным экземплярам полей, присоединенных к типу
    // контента job_post (таких как поле тела), и их удаление по отдельности.
    $instances = field_info_instances('node', 'job_post');
    foreach ($instances as $instance_name => $instance) {
        field_delete_instance($instance);
    }

    // Удаление типа контента.
    node_type_delete('job_post');

    // Удаление информации о полях.
    field_purge_batch(1000);
}

```

Создание файла *.info*

Давайте также создадим файл `job_post.info` и добавим его в папку `job post`:

```

name = Job Post
description = A job posting content type
package = Pro Drupal Development
core = 7.x
files[] = job_post.install
files[] = job_post.module

```

Создание файла *.module*

И, наконец, нам нужен сам файл модуля. Создайте файл по имени `job_post.module` и сохраните его в папке `sites/all/modules/custom/job_posting`. После завершения создания модуля его можно активировать на странице со списками модулей (Modules). Начнем с открывающего дескриптора PHP и краткого комментария.

```
<?php
/**
 * @file
 * Модуль для типа ноды job_post.
 */
```

Предоставление информации о типе ноды

Теперь все готово для добавления хуков в файл `job_post.module`. Сначала реализуем хук `hook_node_info()`. Drupal вызывает этот хук в процессе выяснения доступных типов нод. Здесь можно добавить некоторые метаданные о нашем модуле.

```
/**
 * Реализация hook_node_info() для типа job_post.
 */
function job_post_node_info() {
  return array(
    'job_post' => array(
      'name' => t('Job Post'), // Объявление о рабочей вакансии
      'base' => 'job_post',
      'description' => t('Use this content type to post a job.'),
                          // Тип контента для объявлений о вакансиях.
      'has_title' => TRUE,
      'title_label' => t('Job Title'), // Название работы
      'help' => t('Enter the job title, job description,
                  and the name of the company that posted the job'),
                          // Введите название работы, ее описание и название компании,
                          // поместившей объявление
    ),
  );
}
```

В одном модуле могут быть определены несколько типов нод, поэтому возвращаемое значение должно быть массивом. Ниже приведен список значений метаданных, которые могут быть указаны в хуке `node_info()`.

- `name`. Читательное для человека имя типа ноды. Является обязательным.
- `base`. Базовая строка для создания обратных вызовов, соответствующих данному типу ноды (т.е. если базовая строка определена как `example_foo`, то при вставке ноды этого типа будет вызвана функция `example_foo_insert`). Такая строка обычно, хотя и не всегда, совпадает с именем модуля. Является обязательной.
- `description`. Краткое описание типа ноды. Является обязательным.
- `help`. Подсказка, выводимая пользователю при создании ноды данного типа. Не является обязательной (по умолчанию '').
- `has_title`. Логическое значение, указывающее, имеет ли данный тип ноды поле заголовка. Не является обязательным (по умолчанию TRUE).
- `title_label`. Метка для поля заголовка для данного типа контента. Не является обязательной (по умолчанию содержит строку Title).
- `locked`. Булевское значение, указывающее, может ли администратор изменить машинное имя данного типа. FALSE — может изменять (не заблокировано), TRUE — имя не изменяемое (заблокировано). Не является обязательным (по умолчанию TRUE).

На заметку! Внутреннее поле имени `name`, упомянутое в этом списке (`base`), используется для создания URL-адреса ссылок **Add new content** (Добавить новый контент). Например, в качестве внутреннего имени нашего типа ноды мы используем строку `job_post` (ключ возвращаемого массива), поэтому для создания нового объявления о вакансии пользователи будут направлены по адресу `http://example.com/?q=node/add/job_post`. Обычно не стоит делать это значение изменяемым (`locked = FALSE`). Внутреннее имя хранится в столбце `type` таблиц `node` и `node_revisions`.

Изменение обратного вызова меню

Для реализации хука `hook_menu()` не обязательно наличие ссылки на странице **Add new content** (Добавить новый контент). Среда Drupal автоматически распознает новый тип контента и добавляет его на страницу `http://example.com/?q=node/add`, как показано на рис. 7.2. Прямая ссылка на форму отправки ноды будет выглядеть как `http://example.com/?q=node/add/job_post`. Имя и описание берутся из значений, определенных в `job_post_node_info()`.



Рис. 7.2. Тип контента, появившийся на странице `http://example.com/?q=node/add`

Если прямая ссылка не нужна, ее можно удалить в хуке `hook_menu_alter()`. Например, следующий код удалит страницу для всех пользователей, у которых нет права на администрирование нод:

```
/**
 * Реализация of hook_menu_alter().
 */
function job_post_menu_alter(&$callbacks) {
  // Если у пользователя нет права администрирования нод, элемент меню
  // job_post деактивируется: в его обратный вызов доступа заносится FALSE.

  if (!user_access('administer nodes')) {
    $callbacks['node/add/job_post']['access callback'] = FALSE;

    // Необходимо сбросить аргументы доступа, иначе Drupal использует
    // user_access() в качестве стандартного обратного вызова доступа.
    unset($callbacks['node/add/job_post']['access arguments']);
  }
}
```

Определение прав, специфичных для данного типа ноды, с помощью `hook_permission()`

Обычно в права для типов нод, определенных в модуле, входит возможность создавать ноду данного типа, изменять созданную ноду и изменять любую ноду заданного типа. Они определяются в коде `hook_permission()` как `create job_post`, `edit own job_post`, `edit any job_post` и т.д. Однако их еще понадобится определить в модуле. Сейчас мы создадим их с помощью хука `hook_permission()`:

```
/**
 * Реализация hook_permission().
 */
function job_post_permission() {
  return array(
    'create job post' => array(
      'title' => t('Create a job post'),
      // Создание объявления о вакансии
      'description' => t('Create a job post'),
      // Создание объявления о вакансии
    ),
    'edit own job post' => array(
      'title' => t('Edit own job post'),
      // Редактирование собственного объявления о вакансии
      'description' => t('Edit your own job posting'),
      // Редактирование собственного объявления о вакансии
    ),
    'edit any job post' => array(
      'title' => t('Edit any job post'),
      // Редактирование любого объявления о вакансии
      'description' => t('Edit any job posting'),
      // Редактирование любого объявления о вакансии
    ),
    'delete own job post' => array(
      'title' => t('Delete own job post'),
      // Удаление собственного объявления о вакансии
      'description' => t('Delete own job posting'),
      // Удаление собственного объявления о вакансии
    ),
    'delete any job post' => array(
      'title' => t('Delete any job post'),
      // Удаление любого объявления о вакансии
      'description' => t('Delete any job posting'),
      // Удаление любого объявления о вакансии
    ),
  );
}
```

Если теперь на странице People (Люди) перейти на вкладку Permissions (Права доступа), только что определенные права будут там, и их можно использовать для назначения ролям пользователей.

Ограничение доступа к типу ноды с помощью `hook_access()`

Мы определили права доступа в хуке `hook_perm()`, но как они реализуются? Модули нод могут ограничить доступ к определенным в них типам нод с помощью хука `hook_access()`. Суперпользователь (пользователь с идентификатором 1) может обойти все проверки доступа, поэтому в данном случае этот хук не вызывается. Если данный

хук не определен для нашего типа ноды, все проверки доступа завершатся неудачей, и, значит, только суперпользователь и пользователи с правами администрирования нод могут создавать, редактировать и удалять контент этого типа.

```
/**
 * Реализация hook_access().
 */
function job_access($op, $node, $account) {
  $is_author = $account->uid == $node->uid;
  switch ($op) {
    case 'create':
      // Разрешить, если у роли пользователя есть право создания объявления.
      if (user_access('create job', $account)) {
        return NODE_ACCESS_ALLOW;
      }

    case 'update':
      // Разрешить, если у роли пользователя есть право правки своего
      // объявления и пользователь является автором, либо если у роли
      // пользователя есть право редактирования любого объявления.
      if (user_access('edit own job', $account) && $is_author ||
          user_access('edit any job', $account)) {
        return NODE_ACCESS_ALLOW;
      }

    case 'delete':
      // Разрешить, если у роли пользователя есть право удаления своего
      // объявления и пользователь является автором, либо если у роли
      // пользователя есть право удаления любого объявления.
      if (user_access('delete own job', $account) && $is_author ||
          user_access('delete any job', $account)) {
        return NODE_ACCESS_ALLOW;
      }
  }
}
```

Эта функция позволяет пользователям создавать ноды объявлений о вакансиях, если у их ролей есть право создания объявления о вакансии. Они могут изменять объявления, если у их ролей имеется право редактирования собственного объявления, и они являются авторами, либо у них есть право редактирования любого объявления. Пользователи с правами удаления собственных объявлений могут удалять свои объявления, а пользователи с правами удаления любых объявлений могут удалять любые ноды типа job post.

Еще одно значение \$op, которое передается в функцию hook_node_access() — это view, которое позволяет управлять тем, кто может просматривать данную ноду. Однако учтите, что hook_node_access() вызывается лишь для страниц с отображением одной ноды. Хук hook_node_access() не предотвратит просмотр ноды в представлении тизера на странице со списком нескольких нод. Можно, конечно, поработать и с другими хуками и непосредственно манипулировать значением \$node->teaser, но это неудачный подход. Лучшим решением будет использовать хук hook_node_grants(), о котором будет рассказано чуть ниже.

Настройка формы для нашего типа ноды

Итак, у нас имеются метаданные и права доступа, определенные для нового типа ноды. Теперь нужно создать форму ноды, чтобы пользователи могли вводить объяв-

ления о вакансиях. Для этого понадобится реализовать хук `hook_form()`. В Drupal имеется стандартная форма ноды, содержащая заголовок, тело и все дополнительные поля, которые были определены. Для типа контента объявлений о вакансиях такая форма вполне годится, так что мы используем ее для отображения формы добавления/редактирования.

```
/**
 * Реализация hook_form() на основе стандартной формы.
 */
function job_post_form($node, $form_state) {
  return node_content_form($node, $form_state);
}
```

На заметку! Если вы не знакомы с API-интерфейсом форм, обратитесь к главе 10.

Если вы являетесь администратором сайта и активировали данный модуль, можете теперь перейти на страницу Add content⇒Job Post (Добавить контент⇒Объявление о вакансии) и просмотреть только что созданную форму (рис. 7.3).

Рис. 7.3. Форма для отправки объявления о вакансии

При работе с формой ноды, если это не обобщенная форма, модуль ноды выполняет проверку и сохранение всех стандартных полей, о которых известно модулю внутри формы ноды (вроде полей `title` и `body`), а также предоставляет вам (разработчику) хуки для проверки и сохранения всех собственных полей. Рассмотрим эти возможности.

Проверка полей с помощью `hook_validate()`

При отправке ноды нашего типа модуль вызывается через хук `hook_validate()`. Поэтому, когда пользователь отправляет форму для создания или изменения объявления о вакансии, вызов `hook_validate()` найдет функцию `job_post_validate()`, чтобы проверить данные, введенные в специальных полях. После получения этих данных

их можно изменить с помощью `form_set_value()`, а с помощью `form_set_error()` можно установить код ошибки.

```
/**
 * Реализация hook_validate().
 */

function job_post_validate($node) {
  // Принудительное применение требования: название компании
  // должно содержать не менее 2 символов.
  if (isset($node->job_post_company) &&
      strlen($node->job_post_company['und'][0]['value']) < 2) {
    form_set_error('job_post_company',
      t('The company name is too short. It must be at least 2 characters.'),
      // Слишком короткое название компании. Должно быть не менее 2 символов.

      $limit_validation_errors = NULL);
  }
}
```

Кроме того, мы уже определили минимальное количество слов для поля `body field` в хуке `hook_node_info()`, и Drupal выполнит эту проверку самостоятельно. Однако поле `punchline` является дополнительным полем, добавленным к форме типа ноды, поэтому за его проверку (а также за загрузку и сохранение) отвечаете вы.

Сохранение данных с помощью `hook_insert()`

При сохранении новой ноды вызывается хук `hook_insert()`. Здесь перед сохранением можно выполнить любую специальную обработку содержимого ноды. Данный хук вызывается только для модуля, определенного в метаданных типа ноды. Эта информация определена в ключе `base` хука `hook_node_info()` (см. раздел “Предоставление информации о типе ноды” ранее в главе). Например, если ключ `base` содержит значение `job_post`, то вызывается `job_post_insert()`. Если активирован модуль описания книг и создана новая нода типа `book`, то функция `job_post_insert()` *не будет* вызвана; вместо нее будет выполнен вызов `book_insert()`, т.к. в модуле `book.module` тип ноды определен с помощью ключа `base` со значением `book`.

На заметку! Если требуется что-то сделать с нодой другого типа при ее вставке, используйте вызов `hook_nodeapi()` для внедрения в общий процесс отправки ноды. См. раздел “Работа с нодами произвольного типа с помощью `hook_node_XXXX()`” далее в главе.

Ниже приведен текст функции `hook_insert()` для модуля `job_post.module`. Мы будем создавать запись в таблице слежения при каждом создании новой ноды объявления о вакансии.

```
/**
 * Реализация hook_insert().
 */

function job_post_insert($node) {
  // Запись в журнал информации об объявлении.
  watchdog('job post', 'A new job post titled: '.$node->title.' for company: ' .
    $node->job_post_company['und'][0]['value'],
    ' was added by UID: '.$node->uid, $variables = array(),
    WATCHDOG_NOTICE, $link = 'node/'.$node->nid);
}
```

Поддержка актуальности данных с помощью `hook_update()`

Хук `update()` вызывается при редактировании ноды, когда основные данные ноды уже сохранены в базе данных. В этот момент можно записать изменения в связанные таблицы базы данных. Как и `hook_insert()`, этот хук вызывается только для текущего типа ноды. Например, если ключ `module` для типа ноды в `hook_node_info()` содержит значение `job_post`, то осуществляется вызов `job_post_update()`.

```
/**
 * Реализация hook_update().
 */
function job_post_update($node) {
// Информация о размещении объявления для наблюдения.
  watchdog('job post', 'A job post titled: '.$node->title.' for company: ' .
    $node->job_post_company['und'][0]['value'].
    ' was updated by UID: '.$node->uid, $variables = array(),
    WATCHDOG_NOTICE, $link = 'node/'.$node->nid);
}
```

Очистка с помощью `hook_delete()`

Сразу после удаления ноды из базы данных Drupal оповещает об этом модули с помощью вызова `hook_delete()`. Этот хук обычно используется для удаления соответствующей информации из базы данных. Она вызывается только при удалении текущего типа ноды. Если ключ `base` в `hook_node_info()` для данного типа ноды содержит значение `job_post`, то выполняется вызов `job_post_delete()`.

```
/**
 * Реализация hook_delete().
 */
function job_post_delete($node) {
// Запись в журнал информации об объявлении.
  watchdog('job post', 'A job post titled: '.$node->title.' for company: ' .
    $node->job_post_company['und'][0]['value'].
    ' was deleted by UID: '.$node->uid, $variables = array(),
    WATCHDOG_NOTICE, $link = 'node/'.$node->nid);
}
```

Изменение нод нашего типа с помощью `hook_load()`

Еще один полезный хук для нашего модуля `job_post` позволяет добавлять пользовательские атрибуты ноды в объект ноды при его конструировании. Нам нужно добавить в процесс загрузки ноды спонсора объявления о вакансии, чтобы он был доступен другим модулям и уровню тем. Для этого предназначен хук `hook_load()`.

Этот хук вызывается сразу после создания базового объекта ноды и только для загружаемого текущего типа ноды. Если ключ `module` в `hook_node_info()` для заданного типа ноды содержит значение `job_post`, то выполняется вызов `job_post_load()`. В нашем примере мы вставим атрибут ноды `sponsor` и присвоим ему значение, которое впоследствии будет использоваться где-то в другом месте.

```
/**
 * Реализация hook_load().
 */
function job_post_load($nodes) {
// Добавление в ноду нового элемента во время загрузки
// для сохранения информации о спонсоре объявления о вакансии.
  foreach ($nodes as $node) {
    $node->sponsor = "ACME Career Services, Your Source for Drupal Jobs";
  }
  return $node;
}
```

Использование `hook_view()`

Теперь у нас имеется полная система для ввода и редактирования объявлений о вакансиях. Однако спонсоры будут недовольны, т.к. информация о спонсоре хоть и добавляется с помощью `hook_load()`, но не выводится при просмотре объявлений. Это можно сделать с помощью хука `hook_view()`:

```
/**
 * Реализация hook_view().
 */
function job_post_view($node, $view_mode) {
  // Добавление и темизация информации о спонсоре,
  // чтобы она была видна при просмотре объявления.
  if ($view_mode == 'full') {
    $node->content['sponsor'] = array(
      '#markup' => theme('sponsor',
        array('sponsor' => $node->sponsor, 'sponsor_id' => $node_nid)
      ),
      '#weight' => 100,
    );
  }
  return $node;
}
```

Здесь форматирование информации о спонсоре выделено в отдельную функцию темы, чтобы ее можно было легко переопределять — так мы облегчим жизнь вечно загруженным системным администраторам, которым придется использовать наш модуль и настраивать внешний вид и поведение выходных данных. Для этого создадим функцию `hook_theme()` с определением того, как модуль будет применять тему к новому полю спонсора. В функции `hook_theme()` мы определим переменные, связанные с полем спонсора и файлом шаблона, который будет использован для отображения информации о спонсоре в ноде.

```
/**
 * Реализация hook_theme().
 */
function job_post_theme() {
  // Определение переменных и шаблона, связанных с полем спонсора.
  // Это поле будет содержать имя спонсора, а значение sponsor_id
  // будет применяться для создания уникального идентификатора CSS.
  return array(
    'sponsor' => array(
      'variables' => array('sponsor' => NULL, 'sponsor_id' => NULL),
      'template' => 'sponsor',
    ),
  );
}
```

И последнее, что нужно сделать в данном процессе — создать файл шаблона для отображения информации о спонсоре. В функции `hook_theme()` значение `sponsor` присваивается атрибуту файла шаблона, поэтому необходимо создать файл `sponsor.tpl.php` в каталоге нашего модуля. Ниже приведено содержимое упомянутого файла:

```
<?php

/**
 * @file
```

```

* Реализация стандартной темы для отображения информации о спонсоре объявления.
*
* Доступные переменные:
* $sponsor_id — идентификатор ноды, связанный с объявлением
* $sponsor    — имя спонсора объявления
*/
?>
<div id="sponsor-<?php print $sponsor_id ?>" class="sponsor">
  <div class="sponsor-title">
    <h2>Sponsored by</h2>
  </div>
  <div class="sponsored-by-message">
    This job posting was sponsored by: <?php print $sponsor; ?>
  </div>
</div>

```

Чтобы среда Drupal обратилась к нашему хуку темы, понадобится очистить кэшированный реестр тем. Кэш можно очистить с помощью модуля `devel.module` или со страницы Modules (Модули). Теперь у нас имеется полностью готовая система ввода и просмотра объявлений о вакансиях. Попробуйте ввести несколько объявлений и просмотреть их. Вы увидите объявления в простом формате, как показано на рис. 7.4 и 7.5.

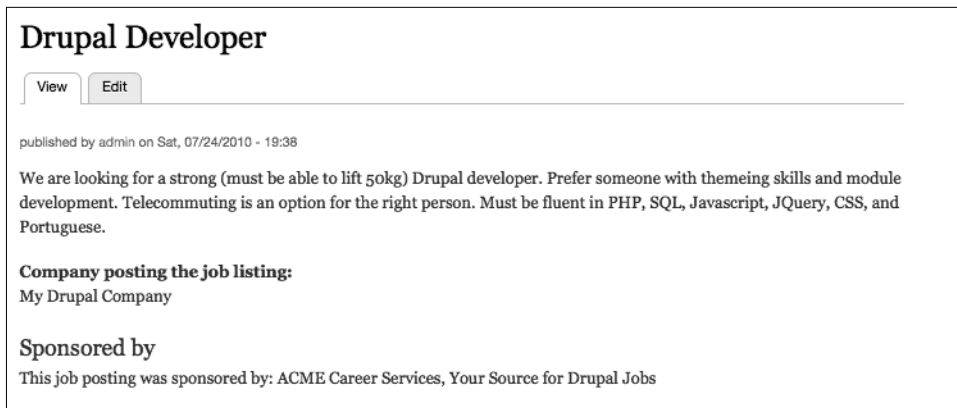


Рис. 7.4. Простая тема для ноды объявления о вакансии

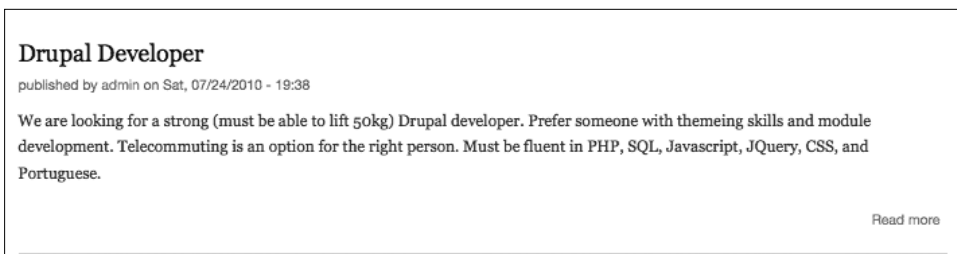


Рис. 7.5. Информация о спонсоре не видна при просмотре в представлении тизера¹

¹ Тизер — рекламное сообщение, построенное в виде загадки и содержащее часть информации о товаре. Обычно появляется на раннем этапе продвижения товара для создания интриги вокруг него.

Работа с нодами произвольного типа с помощью `hook_node_xxxxx()`

Хуки, описанные выше, вызываются только на основе базового ключа из реализации модуля `hook_node_info()`. Когда Drupal видит тип ноды `blog`, выполняется вызов `blog_load()`. Но что если нужно добавить какую-то информацию в каждую ноду, независимо от ее типа? Хуки, с которыми мы знакомы, не способны на это; поэтому необходим исключительно мощный инструмент.

Хуки `node_xxxxx` дают возможность модулям реагировать на различные действия во время жизненного цикла любой ноды. Обычно они вызываются из `node.module` сразу после обратного вызова для конкретного типа ноды. Ниже приведен список первичных функций-хуков `node_xxxxx`.

- `hook_node_insert($node)`. Реагирует на создание новой ноды.
- `hook_node_load($node, $types)`. Действует на ноды, загружаемые из базы данных. `$nodes` — ассоциативный массив загружаемых нод (ключами являются идентификаторы нод), `$types` — массив загружаемых типов нод.
- `hook_node_update($node)`. Реагирует на обновление ноды.
- `hook_node_delete($node)`. Реагирует на удаление ноды.
- `hook_node_view($node, $view_mode)`. Действует на отображаемые ноды, где `$view_mode` определяет режим отображения ноды — т.е. полный или в виде тизера.
- `hook_node_prepare($node)`. Действует на ноды перед их отображением в форме добавления/редактирования.
- `hook_node_presave($node)`. Действует на вставляемые или изменяемые ноды.
- `hook_node_access($node, $op, $account)`. Управляет доступом к ноды, где `$op` — тип выполняемой операции (например, вставка, обновление, просмотр, удаление), а `$account` — учетная запись пользователя, выполняющего операцию.
- `hook_node_grants_alter(&$grants, $account, $op)`. Изменяет полномочия доступа пользователя при попытке просмотра, редактирования или удаления ноды.

Порядок вызова хуков при выводе страницы ноды `http://example.com/?q=node/3` показан на рис. 7.6.

Хранение нод

Ноды хранятся в базе данных в виде отдельных частей. Таблица `node` содержит большинство метаданных, описывающих ноду. Таблица `node_revisions` содержит тело ноды и ее тизер, а также информацию о ревизии. И, как было показано в примере `job_post.module`, другие ноды могут добавлять данные к этой ноды во время ее загрузки и сохранять любые необходимые данные в собственных таблицах.

Объект ноды, содержащий наиболее распространенные атрибуты, показан на рис. 7.7. Таблица, созданная с помощью API-интерфейса полей для публикации объявлений, отличается от основной таблицы нод. В зависимости от того, какие другие модули активированы, объекты нод в вашей инсталляции Drupal могут содержать большее или меньшее количество свойств.

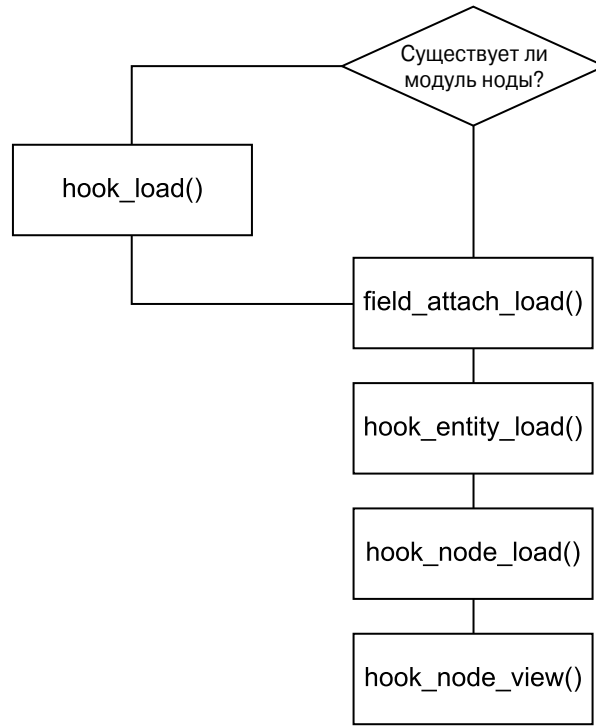


Рис. 7.6. Последовательность выполнения при отображении страницы ноды

Создание типа ноды с помощью собственных типов контента

Способ создания модуля ноды, продемонстрированный в примере с `job_post.module`, обеспечивает исключительные возможности по управлению и производительность, однако он несколько утомителен. А хорошо бы создавать новый тип ноды вообще без программирования! Для этого и предназначены собственные типы контента из ядра Drupal.

Новые типы контента (такие как `job_post`) можно добавлять с помощью интерфейса администрирования: **Structure**⇒**Content types** (Структура⇒Типы контента). Если модуль `job_post.module` уже активен, выберите для типа ноды другое имя, чтобы не было конфликта в пространстве имен. В примере с `job_post.module` нужны три поля: заголовок вакансии, описание вакансии (тело ноды) и название компании, предлагающей работу. В модуле `job_post` нужно было вручную добавить поле тела и название компании. Собственные типы контента из ядра Drupal позволяют устранить все программирование и просто создать ноду с помощью пользовательского интерфейса. Ядро Drupal берет на себя решение всех задач: создание таблиц, вставка, обновление, удаление, управление доступом и просмотр нод.

Ограничение доступа к нодам

Существует несколько способов ограничения доступа к нодам. Мы уже видели, как можно ограничить доступ к типу ноды с помощью хука `hook_access()` и прав доступа, определенных с помощью хука `hook_perm()`. Но Drupal предоставляет намного более мощные инструменты управления доступом — таблицу `node_access` и еще два хука доступа `hook_node_grants()` и `hook_node_access_records()`.

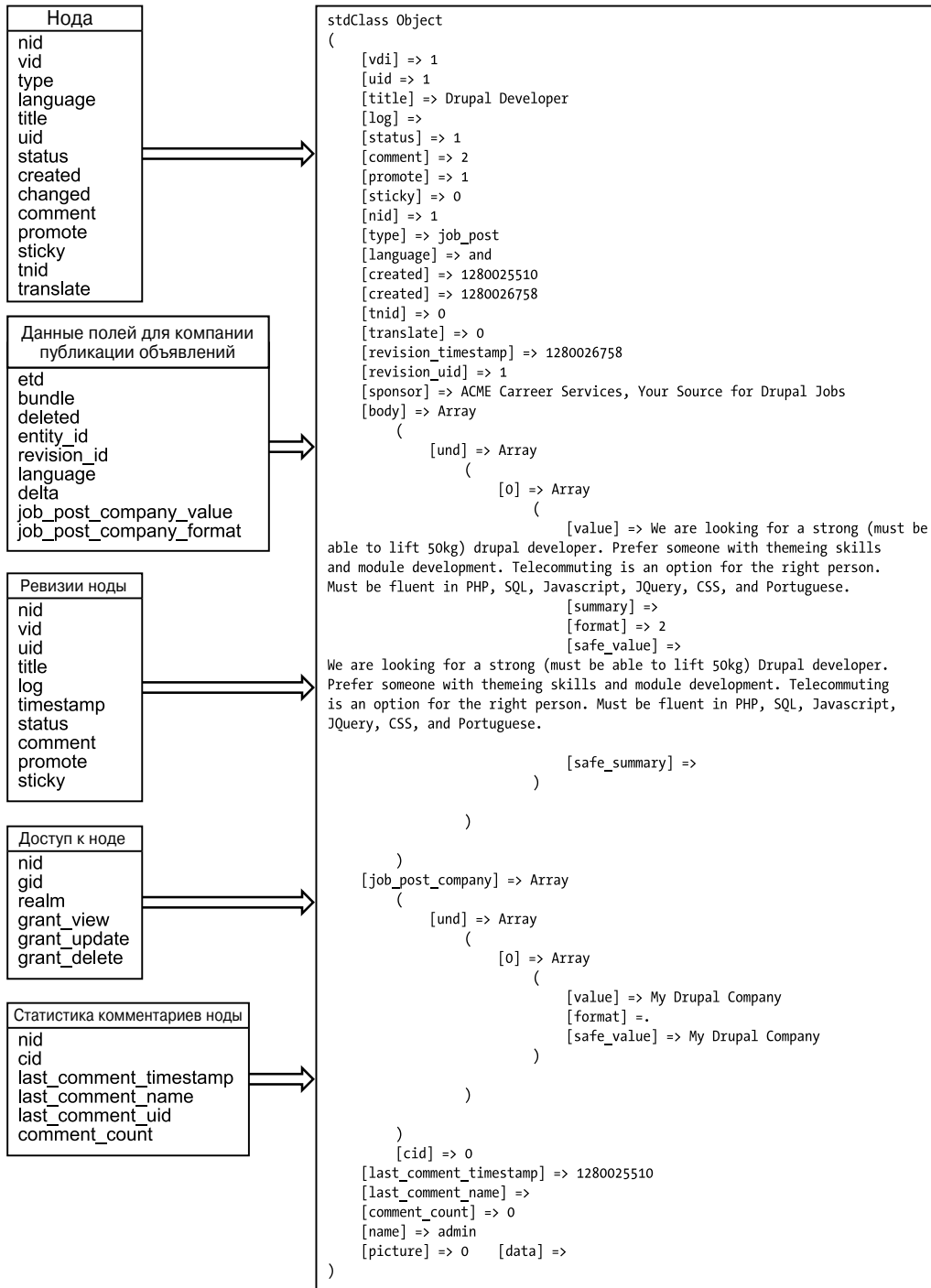


Рис. 7.7. Объект ноды

При первоначальной установке Drupal в таблицу `node_access` заносится одна запись, которая отключает механизм доступа к ноду. Эта часть Drupal включается в работу только при активации модуля, использующего механизм доступа к ноду. Функция `node_access_rebuild()` в файле `modules/node/node.module` отслеживает, какие модули доступа к ноду активированы, и если они отключены, она восстанавливает запись по умолчанию, которая приведена в табл. 7.2.

Таблица 7.2. Запись по умолчанию для таблицы `node_access`

<code>nid</code>	<code>gid</code>	<code>realm</code>	<code>grant_view</code>	<code>grant_update</code>	<code>grant_delete</code>
0	0	all	1	0	0

В общем случае, если используется модуль доступа к ноду (которая модифицирует таблицу `node_access`), Drupal будет запрещать доступ к ноду, пока модуль доступа к этой ноду не вставит в таблицу `node_access` строку, определяющую способ обеспечения доступа.

Определение разрешений ноды

Есть три основных права выполнения действий с нодами: просмотр, обновление и удаление. Перед выполнением любой из этих операций модуль, реализующий тип ноды, сначала обращается к реализации `node_access()`. Если этот модуль не проясняет, разрешен ли доступ (т.е. возвращает `NULL`, а не `TRUE` или `FALSE`), Drupal опрашивает все модули, которым нужен доступ к ноду, чтобы определить, разрешено ли проведение данной операции. Они делают это, выдавая в ответ на вызовы `hook_node_grants()` список идентификаторов разрешений (`grant`) для каждой области, связанной с текущим пользователем.

Что такое область

Область (`realm`) — это произвольная строка, которая позволяет нескольким модулям доступа к ноду разделять таблицу `node_access`. Например, `acl.module` — добавленный модуль, который управляет доступом к ноду с помощью списков управления доступом (Access Control List — ACL). Его областью является `acl`. Еще один дополнительный модуль — `taxonomy_access.module`, который ограничивает доступ к нодам на основе таксонометрических терминов. Он использует область `term_access`. То есть область — это то, что идентифицирует пространство модуля в таблице `node_access`, вроде пространства имен. Когда модуль получает запрос на идентификаторы разрешения, он делает это для определяемой модулем области.

Что такое идентификатор разрешения

Идентификатор разрешения (`grant ID`) — это идентификатор, который предоставляет информацию о правах доступа к ноду для данной области. Например, модуль доступа к ноду — такой как `forum_access.module`, который управляет доступом к нодам типа `forum` на основании ролей пользователей — может использовать идентификаторы ролей в качестве идентификаторов разрешения. Модуль доступа, который управляет доступом к нодам на основе почтовых индексов, мог бы применять в качестве идентификаторов разрешения почтовые индексы. В любом случае это должно быть нечто такое, что каким-то образом определяет пользователя: назначена ли пользователю эта роль? Содержит ли адрес этого пользователя индекс 12345? Включен ли пользователь в данный список управления доступом? Оформил ли пользователь подписку более года назад?

Каждый идентификатор разрешения означает что-то особое для модуля доступа к ноду, который предоставляет идентификаторы разрешения для области, содержащей этот идентификатор разрешения. Однако простое наличие в таблице `node_access` строки, содержащей идентификатор разрешения, уже разрешает доступ, а тип доступа определяется значением 1 в столбце `grant_view`, `grant_update` или `grant_delete`.

Идентификаторы разрешения вставляются в таблицу `node_access` при сохранении ноды. Каждому модулю, в котором реализован хук `hook_node_access_records()`, передается объект ноды. Модуль должен проверить ноду и либо просто завершить работу (если он не будет управлять доступом для этой ноды), либо вернуть массив идентификаторов разрешения для вставки в таблицу `node_access`. Эти разрешения одновременно вставляются с помощью вызова `node_access_acquire_grants()`. Ниже приведен пример из модуля `node.module`. В нем хук `hook_node_access_records()` проверяет, является ли нода приватной; если да, то разрешение дается только на просмотр. Второе разрешение проверяет, является ли пользователь автором ноды; если да, разрешается все — просмотр, обновление и удаление.

```
function hook_node_access_records($node) {
  if (node_access_example_disabling()) {
    return;
  }

  // Проверка выполняется только для нод, которые помечены как приватные.
  // Остальные ноды считаются обычными и игнорируются.
  if ($node->private) {
    $grants = array();
    $grants[] = array(
      'realm' => 'example',
      'gid' => 1,
      'grant_view' => 1,
      'grant_update' => 0,
      'grant_delete' => 0,
      'priority' => 0,
    );

    // В массиве example_author GID равен UID, т.е. существует
    // весьма много групп, содержащих только одного пользователя.
    $grants[] = array(
      'realm' => 'example_author',
      'gid' => $node->uid,
      'grant_view' => 1,
      'grant_update' => 1,
      'grant_delete' => 1,
      'priority' => 0,
    );
    return $grants;
  }
}
```

Процесс доступа к ноду

Перед выполнением действия с нодой среда Drupal выполняет процесс, представленный на рис. 7.8.

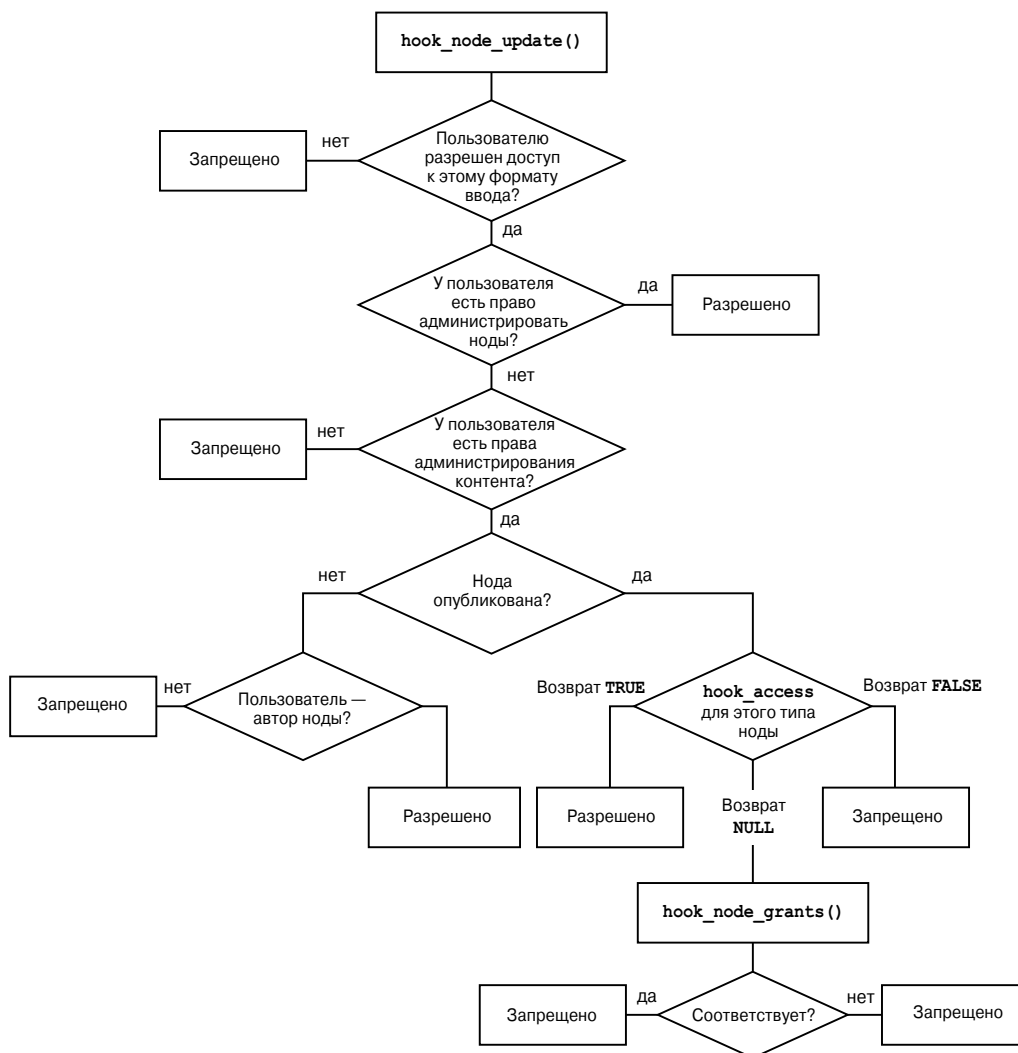


Рис. 7.8. Определение прав доступа к конкретной ноде

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Знать, что такое нода, и какими бывают типы нод.
- Писать модули, которые создают типы нод.
- Уметь вмешиваться в процесс создания ноды, ее сохранения, загрузки и т.д.
- Знать, как определяются права доступа к нодам.

ГЛАВА 8

Работа с полями

Поле — популярный компонент Drupal, используемый для хранения значений, которые введены пользователем или сгенерированы модулем. Информация из полей проверяется, хранится в базе данных и может быть извлечена из базы и выведена на веб-сайте. Примерами полей могут служить пользовательское имя, почтовый адрес, номер телефона, цена, пара абзацев с описанием события, дата или любой другой фрагмент информации, который можно себе представить.

В Drupal 7 способ создания полей существенно изменился из-за добавления в ядро API-интерфейса для работы с полями — Field API. Утомительная ранее задача определения таблиц и написания кода для проверки, хранения, выборки и вывода информации уровня полей сейчас осуществляется с помощью набора интерфейсов Field API. Многие из возможностей уровня полей, добавленные в ядро Drupal 7, произошли из модулей набора создания контента (Content Construction Kit — ССК). Эти модули предоставляли пользовательский интерфейс для создания полей в предыдущих версиях Drupal.

В данной главе мы рассмотрим стандартные функции в стиле ССК, встроенные в ядро Drupal 7, способы расширения этих функций за счет добавления новых типов полей, которые любой администратор сайта может присоединить к типу контента, и в конце — использование Field API в модуле для создания нового типа контента с несколькими различными типами полей.

Создание типов контента

Одним из впечатляющих аспектов Drupal является возможность создания произвольного типа контента, который определен в качестве инфраструктуры для создания ноды. Типы контента обычно содержат, по крайней мере, поле заголовка и поле тела, а также несколько других необязательных полей, которые применяются для ввода структурированной информации. Примером пользовательского типа контента является событие, в котором имеются поля для ввода, хранения и вывода информации вроде имени события (заголовка), описания (тела), даты и времени и его местоположения. Давайте создадим новый тип контента — событие. Для этого перейдите на страницу [Structure⇒Content Types](#) (Структура⇒Типы контента) и щелкните на ссылке [Add content type](#) (Добавить тип контента). Создать новый тип контента относительно несложно: достаточно ввести нужные значения для имени типа контента (в данном случае — Event) и краткого описания типа контента; кроме того, можно переопределить заголовок ноды Event. В нашем примере мы изменим этот заголовок с ничего не значащего Title (Заголовок) на Event Title (Заголовок события), как показано на рис. 8.1.

Content types

Individual content types can have different fields, behaviors, and permissions assigned to them.

Name *
 Event Machine name: event [Edit]

The human-readable name of this content type. This text will be displayed as part of the list on the *Add new content* page. It is recommended that this name begin with a capital letter and contain only letters, numbers, and spaces. This name must be unique.

Description
 A content type used to store and display information about events

Describe this content type. The text will be displayed on the *Add new content* page.

Submission form settings
 Event Title

Title field label *
 Event Title

Publishing options
 Published, Promoted to front page

Display settings
 Display author and date information.

Comment settings
 Open, Threading, 50 comments per page

Menu settings

Preview before submitting
☐ Disabled
☒ Optional
☐ Required

Explanation or submission guidelines

This text will be displayed at the top of the page when creating or editing content of this type.

Save content type Save and add fields

Рис. 8.1. Главная страница для создания типа контента

После определения значений параметров для формы отправки данных необходимо изменить параметры публикации, чтобы они соответствовали специальным требованиям вашего сайта. Перейдите на вкладку **Publishing options** (Параметры публикации) и отметьте или снимите отметку с флажков, которые нужно применить к событиям. В нашем случае снимите отметку с флажка **Promoted to front page** (Выдвинут на переднюю страницу), который по умолчанию отмечен.

В форме **Display settings** (Параметры отображения) снимите отметку с флажка, который указывает выводить информацию об авторе при выводе события. В нашем случае информация об авторе события и времени его публикации не нужна.

Форма **Comment settings** (Параметры комментирования) управляет отображением комментариев для типа контента. Для типа **Event** комментарии не нужны, поэтому для параметра **Default comment setting for new content** (Настройка комментирования по умолчанию для нового контента) выбираем вариант **Hidden** (Скрытый).

В форме **Menu settings** (Параметры меню) снимите отметку с флажка **Main menu** (Главное меню): авторы контента не смогут назначать события пунктам меню. Чтобы указать места для вывода событий, будем вставлять их в представления на тех страницах, где они необходимы.

После завершения такого обширного конфигурирования типа контента все готово для сохранения типа контента `Event` и перехода к следующему шагу — добавлению полей (рис. 8.2). Если щелкнуть на кнопке `Save and add fields` (Сохранить и добавить поля), то Drupal выведет страницу, где к типу контента `Event` можно добавлять новые поля.

LABEL	WEIGHT	PARENT	NAME	FIELD	WIDGET	OPERATIONS
Event Title	-5	<none>	title	Node module element		
Body	-4	<none>	body	Long text and summary	Text area with a summary	edit delete

Add new field

field_ -3 <none> - Select a field type - - Select a widget -
Field name (a-z, 0-9, _)
Type of data to store. Form element to edit the data.

Add existing field

- Select an existing field - - Select a widget -
Field to share. Form element to edit the data.

Save

Рис. 8.2. Страница для добавления полей

Но прежде чем приступить к добавлению полей, необходимо решить, будут даты выводиться в обычном текстовом поле или в специальном поле даты со всплывающим календарем для выбора даты. Обычно требуется второй вариант, т.к. даты часто применяются для других целей: определение момента занесения элемента контента, форматирование дат с учетом локального формата пользователя или выполнение вычислений с датами. Модуль `Date` (<http://drupal.org/project/date>) предоставляет поле, которое можно задействовать в нашем типе контента `Event` и которое содержит всплывающий календарь для выбора даты. Поэтому до добавления полей нужно установить модуль `Date` с помощью стандартного подхода к установке модулей.

Добавление полей в тип контента

Два дополнительных поля для нашего типа контента `Event` должны содержать место и дату/время возникновения события. Начнем с поля местоположения: введем в поле метки текст `Event Location` (Место возникновения события), в качестве имени поля — `event_location`, а в качестве типа данных для хранения — элемент текстового поля (рис. 8.3).

LABEL	WEIGHT	PARENT	NAME	FIELD	WIDGET	OPERATIONS
Event Title	-5	<none>	title	Node module element		
Body	0	<none>	body	Long text and summary	Text area with a summary	edit delete
Event Location	1	<none>	field_event_location	Text	Text field	

Add new field

Event Location 1 <none> - Select a field type - - Select a widget -
Label Field name (a-z, 0-9, _)
Type of data to store. Form element to edit the data.

Add existing field

- Select an existing field - - Select a widget -
Field to share. Form element to edit the data.

Save

Рис. 8.3. Добавление поля `Event Location` к типу контента `Event`

EVENT LOCATION FIELD SETTINGS

These settings apply to the *Event Location* field everywhere it is used.

Number of values

Maximum number of values users can enter for this field.
 'Unlimited' will provide an 'Add more' button so the users can add as many values as they like.

Maximum length *

The maximum length of the field in characters.

Рис. 8.5. Указание кардинальности и максимального количества символов для поля

FIELD SETTINGS

These settings apply to the *Event Date and Time* field everywhere it is used. These settings impact the way that data is stored in the database and cannot be changed once data has been created.

To Date

Display a matching second date field as a 'To date' . If marked 'Optional' field will be presented but not required. If marked 'Required' the 'To date' will be required if the 'From date' is required or filled in.

Granularity

 Month
 Day
 Hour
 Minute
 Second

Set the date elements to be stored (at least a year is required).

Time zone handling

Select the timezone handling method to be used for this date field.

Рис. 8.6. Страница Field Settings для поля Event Date and Time

После щелчка на кнопке **Save field settings** (Сохранить параметры поля) откроется страница **Event settings** (Параметры события) для поля **Event Date** (Дата события). На этой странице можно переопределить введенную ранее метку; указать, что поле является обязательным (для событий на нашем сайте дата и время обязательны); задать текст подсказки, который выдается под полем; стандартный формат даты в модуле даты; значение по умолчанию, которое будет использоваться, если автор не выберет никакого значения; входной формат, определяющий порядок и формат частей даты во входной форме (например, 08/12/2010 – 08:00:00, или 12/08/2010 – 08:00:00, или Aug 12, 2010 – 08:00:00 и т.д.); количество лет в прошлом и будущем, которое будет выдаваться во всплывающем календаре; а также шаг приращения значения минут (если событие происходит только на границе часа, каждые полчаса или четверть часа, то шаг можно указать равным 60, 30 и 15 соответственно). Здесь также можно задать кардинальность или количество значений, которые допустимо создавать (в наших условиях возможно лишь одно значение), допустимы ли диапазоны “от” и “до”, шаг изменения даты и времени и возможность заново указать, должен ли Drupal преобразовывать введенное время при выводе в зависимости от различных настроек.

В date возникновения события оставьте все значения по умолчанию и сохраните поле, щелкнув на кнопке **Save settings** (Сохранить параметры). Теперь есть все поля, необходимые для типа контента **Event**, и можно приступать к созданию событий с помощью формы создания ноды, показанной на рис. 8.7.

Рис. 8.7. Создание нового события

На заметку! На момент написания этих строк выполнялась существенная переработка модуля **Date**, вызванная изменениями в ядре Drupal. Лучше просмотреть информацию на странице <http://drupal.org/project/date> обо всех изменениях в подходе или формах, используемых для настройки поля даты.

Создание произвольного поля

Drupal 7 поставляется с несколькими обобщенными типами полей, которые можно использовать для самых разных целей. С их помощью можно вводить, сохранять и выводить значения, но бывают случаи, когда стандартные типы полей (табл. 8.1) все-таки не годятся. И вот здесь может пригодиться интерфейс **Field API**, позволяющий создавать произвольные типы полей, которые можно применять в любом типе контента, созданном на вашем сайте.

Интерфейс **Field API** позволяет создавать собственный тип поля для практически любого типа вводимых данных, который только можно себе представить. Возьмем в качестве примера тип поля с сайта Drupal.org (http://api.drupal.org/api/drupal/developer--examples--color_example--color_example.module/7), где определено собственное поле для отображения текста в цвете, заданном параметрами для этого типа поля. Мы используем этот тип поля для ввода и отображения цвета, которым должны быть раскрашены участники события при его посещении.

Таблица 8.1. Стандартные типы полей в ядре Drupal 7

Тип поля	Назначение
Boolean (Булевское)	Поле для ввода значений TRUE/FALSE с помощью флажков или переключателей
Decimal (Десятичное)	Поле для ввода числовых значений, которые могут содержать дробную часть
File (Файл)	Поле для загрузки файла, которое позволяет авторам прикреплять файл к экземпляру типа контента
Float (С плавающей точкой)	Текстовое поле для ввода чисел, которые могут содержать дробную часть
Image (Изображение)	Поле для загрузки изображений, которое позволяет авторам прикреплять изображение к экземпляру типа контента
Integer (Целое)	Текстовое поле для ввода целочисленных значений
List (Список — обычный, числовой или текстовый)	Позволяет создать список выбора (выпадающий или фиксированный), флажки или переключатели для выбора одного или нескольких значений из ряда predetermined значений
Long text (Длинный текст)	Многострочная текстовая область для ввода текста (в отличие от однострочного текстового поля)
Long text and Summary (Длинный текст и резюме)	Многострочная текстовая область для ввода текста и многострочная область резюме для ввода информации
Term Reference (Ссылка на термин)	Поле для выбора таксонометрического термина (терминов)
Text (Текст)	Простое текстовое поле

Сначала понадобится создать в папке `sites/all/modules/custom` новый каталог по имени `color_example`. В этом каталоге создайте новый файл с именем `color_example.info` и поместите в него следующий текст:

```
name = Color Example
description = "Creates a custom field for inputting and displaying text in a
colorful fashion."
package = Pro Drupal Development
core = 7.x
files[] = color_example.module
php = 5.2
```

После этого создайте файл `color_example.module` со следующим содержимым:

```
<?php
/**
 * @file
 * Пример поля, использующего Field API.
 *
 */
```

Сохраните эти файлы и активируйте модуль. Теперь все готово для создания содержимого типа поля RGB.

Сначала нужно вызвать хук `hook_field_info()`, который определяет основные атрибуты нового поля. Мы определим поле как `color_example_rgb()` и назначим новому типу поля метку, описание, стандартный управляющий элемент и стандартный форматер.

```

/**
 * Реализация hook_field_info().
 *
 * Описание для поля.
 */
function color_example_field_info() {
  return array(
    'color_example_rgb' => array(
      'label' => t('Example Color RGB'),
      // Пример RGB-цвета
      'description' => t('Demonstrates a field composed of an RGB color.'),
      // Демонстрация поля, составленного из RGB-цвета
      'default_widget' => 'color_example_3text',
      'default_formatter' => 'color_example_simple_text',
    ),
  );
}

```

Теперь необходимо определить, каким образом данные, введенные в поле, будут храниться в базе данных Drupal. До появления Drupal 7 и Field API нам пришлось бы самостоятельно определять в модулях таблицы и схемы, которые описывали бы наш тип контента и специализированные поля в этом типе контента. В Drupal 7 с Field API эту задачу выполняет функция `hook_field_schema()`. В рассматриваемом примере мы будем хранить семисимвольное поле, представляющее шестнадцатеричный HTML-код цвета для отображения текста на экране — например, содержимое поля `#FF0000` приведет к отображению текста красным цветом. Для этого будет создан один столбец для хранения RGB-значения, введенного администратором сайта при назначении полю типа контента.

```

/**
 * Реализация hook_field_schema().
 */
function color_example_field_schema($field) {
  $columns = array(
    'rgb' => array('type' => 'varchar', 'length' => 7, 'not null' => FALSE),
  );
  $indexes = array(
    'rgb' => array('rgb'),
  );
  return array(
    'columns' => $columns,
    'indexes' => $indexes,
  );
}

```

Теперь нужно обеспечить проверку вводимых пользователем данных с помощью хука `hook_field_validate()`. Мы будем проверять, что пользователь ввел значение в соответствии с правилами HTML, используя функцию `preg_match()`. Необходимо проверить, что первый символ равен `#`, а следующие шесть символов содержат либо цифры, либо буквы от `a` до `f`. Если введенное значение не соответствует этому шаблону, будет выведено сообщение об ошибке.

```

/**
 * Реализация hook_field_validate().
 *
 * Проверка правильности содержимого поля RGB
 * (6 шестнадцатеричных цифр, перед которыми находится #).
 */

```

```
function color_example_field_validate($entity_type, $entity, $field,
$instance, $langcode, $items, &$amp;errors) {
  foreach($items as $delta => $item) {
    if(!empty($item['rgb'])) {
      if(! preg_match('@^[0-9a-f]{6}$@', $item['rgb'])) {
        $errors[$field['field_name']][$langcode][$delta][] = array(
          'error' => 'color_example_invalid',
          'message' => t('Color must be in the HTML format #abcdef.'),
          // Цвет должен быть задан в HTML-формате #abcdef.
        );
      }
    }
  }
}
```

Следующая функция определяет, из чего состоит пустое поле этого типа. В данном случае применяется PHP-функция `empty`, которая возвращает `TRUE` или `FALSE` в зависимости от того, пустое ли поле.

```
/**
 * Реализация hook_field_is_empty().
 */
function color_example_field_is_empty($item, $field) {
  return empty($item['rgb']);
}
```

Форматер поля — это функция, которая определяет, как должно отображаться содержимое поля. Функция `hook_field_formatter_info()` определяет типы форматеров, которые используются в рассматриваемом примере для вывода текста и фона.

```
/**
 * Реализация hook_field_formatter_info().
 */
function color_example_field_formatter_info() {
  return array(
    // Этот форматер просто выводит шестнадцатеричное значение цвета.
    'color_example_simple_text' => array(
      'label' => t('Simple text-based formatter'),
      // Простой текстовый форматер
      'field types' => array('color_example_rgb'),
    ),
    // Этот форматер изменяет фоновый цвет области контента.
    'color_example_color_background' => array(
      'label' => t('Change the background of the output text'),
      // Изменение фона выводимого текста
      'field types' => array('color_example_rgb'),
    ),
  );
}
```

Теперь потребуется создать отображаемые данные для каждого из только что определенных форматеров.

1. Параметр `color_example_simple_text` просто выводит разметку, означающую введенный цвет, и использует внутрискриптовый стиль для задания текстового значения цвета.
2. Параметр `color_example_color_background` делает то же самое, но еще и изменяет фоновый цвет `div.region-content`.

```

/**
 * Реализация hook_field_formatter_view().
 */
function color_example_field_formatter_view($entity_type, $entity, $field,
$instance, $langcode, $items, $display) {
  $element = array();

  switch ($display['type']) {
    // Этот формater просто выводит поле в текстовом виде заданным цветом.
    case 'color_example_simple_text':
      foreach ($items as $delta => $item) {
        $element[$delta]['#markup'] = '<p style="color: ' . $item['rgb'] . '">'
        . t('The color for this event is @code', array('@code' => $item['rgb'])). '</p>';
      }
      break;
    // Этот формater добавляет на страницу CSS-форматирование,
    // изменяя фоновый цвет области .region-content.
    // При наличии нескольких полей в расчет берется последнее.
    case 'color_example_color_background':
      foreach ($items as $delta => $item) {
        drupal_add_css('div.region-content { background-color:' . $item['rgb']
        . '};', array('type' => 'inline') );
        $element[$delta]['#markup'] = '<p>'
        . t('The color for this event has been changed to @code',
        array('@code' => $item['rgb'])). '</p>';
      }
      break;
    }
  }
  return $element;
}

```

Следующий набор функций определяет графический элемент, который будет использоваться для отображения поля на форме редактирования ноды. Для поля RGB мы создадим три различных типа графических элементов, один из которых и будет выбирать администратор.

1. Простое текстовое поле, в котором пользователь введет что-то вроде #ffffff.
2. Поле с тремя текстовыми значениями, в которых будет отдельно указаны значения для красного, зеленого и синего цветов.
3. Графическая палитра для выбора значения.

Для определения этих трех элементов применяется функция `hook_field_widget_info()`:

```

/**
 * Реализация hook_field_widget_info().
 */
function color_example_field_widget_info() {
  return array(
    'color_example_text' => array(
      'label' => t('RGB value as #ffffff'), // RGB-значение в виде #ffffff
      'field types' => array('color_example_rgb'),
    ),
    'color_example_3text' => array(
      'label' => t('RGB text fields'), // Текстовые поля для значений RGB
      'field types' => array('color_example_rgb'),
    ),
  ),
}

```



```

'color_example_colorpicker' => array(
  'label' => t('Color Picker'),          // Палитра для выбора цвета
  'field types' => array('color_example_rgb'),
),
);
}

```

Функция `hook_widget_form()` определяет реальную структуру отображения графических элементов для пользователя. Для имеющихся трех типов элементов задаются три различные формы.

1. `color_example_text` — текстовое поле для ввода HTML-кода цвета (например, #FFFFFF).
2. `color_example_colorpicker` — похож на `color_example_text`, но с добавлением JavaScript-компонента выбора цвета.
3. `color_example_3text` — выводит три текстовых поля для яркости красного, зеленого и синего цвета. Но все же этот тип поля определяет один текстовый столбец, который предназначен для хранения HTML-кода цвета. Для него нужен обработчик проверки, который преобразует три значения из полей красного, зеленого и синего цвета в единый элемент формы.

```

/**
 * Реализация hook_field_widget_form().
 */
function color_example_field_widget_form(&$form, &$form_state, $field,
$instance, $langcode, $items, $delta, $element) {
  $value = isset($items[$delta]['rgb']) ? $items[$delta]['rgb'] : '';
  $element += array(
    '#delta' => $delta,
  );
  $element['rgb'] = array();

  switch ($instance['widget']['type']) {
    case 'color_example_colorpicker':
      $element['rgb'] += array(
        '#suffix' => '<div class="field-example-colorpicker"></div>',
        '#attributes' => array('class' => array('edit-field-example-colorpicker')),
        '#attached' => array(
          // Добавление палитры для выбора цвета.
          'library' => array(
            array('system', 'farbtastic'),
          ),
          // Добавление javascript-кода, который скрывает/отображает палитру.
          'js' => array(drupal_get_path('module', 'color_example')
            . '/color_example.js'),
        ),
      );

      // С этого момента color_example_text и color_example_colorpicker
      // в точности совпадают.
      case 'color_example_text':
        $element['rgb'] += array(
          '#title' => t('Event's RGB Color'),
          '#type' => 'textfield',
          '#default_value' => $value,
          // Немного увеличиваем длину поля, чтобы поместились
          // некоторые сочетания символов.

```

```

        '#size' => 7,
        '#maxlength' => 7,
    );
    break;

case 'color_example_3text':
    // Преобразование значения rgb в r, g и b для #default_value.
    if (isset($items[$delta]['rgb'])) {
        preg_match_all('@..@', substr($items[$delta]['rgb'], 1), $match);
    }
    else {
        $match = array(array());
    }

    // Набор из трех полей для необходимых значений.
    $element += array(
        '#type' => 'fieldset',
        '#element_validate' => array('color_example_3text_validate'),

        // Этот фрагмент нужен, чтобы функция проверки смогла
        // получить внешнее значение.
        '#delta' => $delta,
        '#attached' => array(
            'css' => array(drupal_get_path('module', 'color_example')
. '/color_example.css'),
        ),
    );

    // Создание текстового поля для значений насыщенности Red, Green и Blue.
    foreach (array('r' => t('Red'), 'g' => t('Green'), 'b' => t('Blue'))
as $key => $title) {
        $element[$key] = array(
            '#type' => 'textfield',
            '#title' => $title,
            '#size' => 2,
            '#default_value' => array_shift($match[0]),
            '#attributes' => array('class' => array('rgb-entry')),
            // '#description' => t('The 2-digit hexadecimal representation of the
@color saturation, like "a1" or "ff"', array('@color' => $title)),
        );
    }
    break;

}
return $element;
}

```

Следующая функция определяет проверки для данных, введенных пользователем:

```

/**
 * Проверка отдельных полей и их преобразование в единое текстовое
 * HTML-значение RGB.
 */
function color_example_3text_validate($element, &$amp;$form_state) {
    $delta = $element['#delta'];
    $field = $form_state['field'][$element['#field_name']]
[$element['#language']]['field'];
    $field_name = $field['field_name'];

```

```

if (isset($form_state['values'][$field_name][$element['#language']][$delta])) {
    $values = $form_state['values'][$field_name][$element['#language']][$delta];
    foreach (array('r', 'g', 'b') as $colorfield) {
        $val = hexdec($values[$colorfield]);

        // Если какое-то из полей пусто, устанавливаем пустое значение и выходим.
        if (strlen($values[$colorfield]) == 0) {
            form_set_value($element, array('rgb' => NULL), $form_state);
            return;
        }

        // Если в поле не шестнадцатеричное значение, отбрасываем его.
        if ( (strlen($values[$colorfield]) != 2) || $val < 0 || $val > 255) {
            form_error($element[$colorfield],
                t("Saturation value must be a 2-digit hexadecimal value between 00
and ff."));
            // Значение насыщенности должно быть 2-значным
            // шестнадцатеричным значением от 00 до ff.
        }
    }

    $value = sprintf('%02s%02s%02s', $values['r'], $values['g'], $values['b']);
    form_set_value($element, array('rgb' => $value), $form_state);
}
}

```

Теперь воспользуемся функцией `hook_field_error()` для вывода сообщения об ошибке, когда пользователь ввел что-то неправильно:

```

/**
 * Реализация hook_field_error().
 */
function color_example_field_widget_error($element, $error, $form, &$form_state)
{
    switch ($error['error']) {
        case 'color_example_invalid':
            form_error($element, $error['message']);
            break;
    }
}

```

После этого создадим JavaScript-файл, который выводит палитру выбора цвета для наиболее сложного графического элемента. Создайте в каталоге модуля файл `color_example.js` и поместите в него следующий код:

```

/**
 * @file
 * Код Javascript для примера с цветом.
 */
/**
 * Палитра выбора цвета для наиболее сложного графического элемента.
 */
(function ($) {
    Drupal.behaviors.color_example_colorpicker = {
        attach: function(context) {
            $(".edit-field-example-colorpicker").live("focus", function(event) {
                var edit_field = this;
                var picker = $(this).closest('tr').find(".field-example-colorpicker");
            });
        }
    };
}

```

```

    // Скрытие всех элементов выбора цвета, кроме данного.
    $(".field-example-colorpicker").hide();
    $(picker).show();
    $.farbtastic(picker, function(color) {
        edit_field.value = color;
    }).setColor(edit_field.value);
  });
}
})(jQuery);

```

И последний файл, необходимый для модуля цвета — это файл CSS. Создайте новый файл по имени `color_example.css` и поместите в него следующий CSS-код:

```

/**
 * @file
 * CSS для примера с цветом.
 */
div.form-item table .form-type-textfield,
div.form-item table .form-type-textfield * {
  display: inline-block;
}

```

После сохранения модуля поле готово для добавления типа контента. Добавьте поле Event Color (Цвет события) к событию: перейдите на страницу Structure⇒Content Types (Структура⇒Типы контента), а затем на вкладку Manage Fields (Управление полями) для типа контента Event, как показано на рис. 8.8.

Add new field					
Event Color	3	<none>	field_event_color	Example Color RGB	RGB value as #ffffff
Label			Field name (a-z, 0-9, _)	Type of data to store.	Form element to edit the data.

Рис. 8.8. Добавление нового поля Event Color

После щелчка на кнопке Save (Сохранить) вы увидите следующую страницу параметров поля, где будет видно, что для поля Event Color не заданы никакие параметры (например, максимальная длина). Щелкните на кнопке Save settings (Сохранить параметры), чтобы перейти на страницу сводки параметров (рис. 8.9). На этой форме введите текст подсказки, который будет отображаться на форме под полем, и цвет по умолчанию, который будет использоваться при выводе формы редактирования ноды. Щелкните на кнопке Save settings, чтобы завершить процесс добавления поля на форму редактирования ноды.

После добавления поля к типу контента Event можно приступить к тестированию. Перейдите на страницу Add content⇒Event (Добавить контент⇒Событие), и вы увидите на форме настройки новое поле Event's RGB Color (RGB-цвет события) со значением по умолчанию (рис. 8.10).

После сохранения события новое поле отображается с учетом форматирования, определенного в функциях формatera модуля. На рис. 8.11 показан текст, выведенный цветом, который определен при создании ноды — в данном случае красным.

До этого момента демонстрировалось только применение стандартных типов полей Drupal для нового типа контента и создание типа поля, который можно добавить в тип контента. А теперь посмотрим, как можно добавлять поля в модуле программным образом с помощью Field API.

EVENT SETTINGS

These settings apply only to the *Event Color* field when used in the *Event* type.

Label *

Event Color

☐ Required field

Help text

Enter the Event Color

Instructions to present to the user below this field on the editing form.
 Allowed HTML tags: <a> <big> <code> <del> <i> <ins> <pre> <q> <small> <sub> <sup> <tt> <p>

DEFAULT VALUE

The default value for this field, used when creating new content.

#ff0000

EVENT COLOR FIELD SETTINGS

These settings apply to the *Event Color* field everywhere it is used.

Number of values

1

Maximum number of values users can enter for this field.
 "Unlimited" will provide an "Add more" button so the users can add as many values as they like.

Save settings

Рис. 8.9. Установка параметров конфигурации для нового поля Event Color

Event Location *

Enter details about the location of the Event.

Event Date and Time *

08/16/2010 05:10

Format: 08/16/2010 Format: 05:10

Event's RGB Color

#ff0000

Рис. 8.10. Новое поле Event's RGB Color на форме редактирования ноды

Home

A Red Event View Edit Track

This event is for all of you red fans. Make sure you wear something red!

Event Location:
The Arena

Event Date and Time:
Mon, 08/16/2010 - 05:19

Event Color:
The color for this event is #ff0000

Рис. 8.11. Новое поле выведено цветом, который указал автор

Программное добавление полей

Интерфейс Field API можно использовать для программного добавления полей к типу контента или типу ноды. В следующем примере будет показано применение Field API для добавления нового поля к типу контента, созданного с помощью модуля. Модуль Job Post создает тип контента, который расширяет традиционные ноды (заголовок и тело) за счет добавления нового поля, в котором хранится название компании-спонсора объявления о вакансии. Добавление полей производится в файле `.install` модуля, в функции `hook_install()` или `hook_update()`.

На первом шаге в функции `hook_install()` добавляется поле тела для нового типа контента Job Post. По умолчанию тип контента, созданный в модуле, содержит только поле заголовка, и необходимо явно добавить в него поле тела. В файле `node.module` определяется функция по имени `node_add_body_field()`, которая добавляет к типу контента Job Post стандартное поле тела. Теперь нужно добавить определения всех необходимых полей к нашему типу контента. В данном случае требуется добавить только одно поле с именем `job_post_companies`. Если бы понадобилось добавить несколько полей, то дополнительные определения полей можно было бы создать в функции `_job_post_installed_fields()`. Для определения нового поля достаточно указать его имя, метку и тип поля, после чего можно будет создавать экземпляры полей с помощью функции `field_create_instance()`. Данная функция выполняет все необходимые действия по созданию механизма хранения введенных пользователем значений в базе данных. При этом нет необходимости определять таблицы в базе данных: интерфейс Field API делает это самостоятельно. После инсталляции модуля поля уже находятся на форме редактирования ноды, готовые для использования в модуле. Более подробная информация по применению и темизации данных из самостоятельно созданных полей приведена в главе 7.

```
<?php
/**
 * @file
 * Установочный файл для модуля Job Post.
 */

/**
 * Реализация hook_install().
 *
 * - Добавление поля тела.
 * - Настройка поля тела.
 * - Создание поля с названием компании.
 */
function job_post_install() {
  node_types_rebuild();
  $types = node_type_get_types();
  node_add_body_field($types['job_post']);

  // Загрузка определения экземпляра для тела типа контента.
  $body_instance = field_info_instance('node', 'body', 'job_post');

  // Добавление режима отображения job_post_list в отображение экземпляра тела.
  $body_instance['type'] = 'text_summary_or_trimmed';

  // Сохранение изменений в экземпляре поля тела.
  field_update_instance($body_instance);
```

```

// Создание всех полей, добавленных в тип контента.
foreach (_job_post_installed_fields() as $field) {
    field_create_field($field);
}

// Создание всех рабочих наших полей.
foreach (_job_post_installed_instances() as $instance) {
    $instance['entity_type'] = 'node';
    $instance['bundle'] = 'job_post';
    field_create_instance($instance);
}
}

/**
 * Возврат структурированного массива с определениями полей,
 * созданных данным типом контента.
 */
function _job_post_installed_fields() {
    $t = get_t();
    return array(
        'job_post_company' => array(
            'field_name'      => 'job_post_company',
            'label'           => $t('Company posting the job listing'),
            'type'            => 'text',
        ),
    );
}

/**
 * Возврат структурированного массива с определениями
 * экземпляров данного типа контента.
 */
function _job_post_installed_instances() {
    $t = get_t();
    return array(
        'job_post_company' => array(
            'field_name'      => 'job_post_company',
            'type'           => 'text',
            'label'          => $t('Company posting the job listing'),
            'widget'         => array(
                'type'        => 'text_textfield',
            ),
            'display' => array(
                'job_post_list' => array(
                    'label'      => $t('Company posting the job listing'),
                    'type'       => 'text',
                ),
            ),
        ),
    );
}

/**
 * Реализация hook_uninstall().
 */

```

```

function job_post_uninstall() {
  // Сбор всего контента, который может быть создан при активации модуля.
  $sql = 'SELECT nid FROM {node} n WHERE n.type = :type';
  $result = db_query($sql, array(':type' => 'job_post'));
  $nids = array();
  foreach ($result as $row) {
    $nids[] = $row->nid;
  }

  // Одновременное удаление всех нод.
  node_delete_multiple($nids);

  // Проход в цикле по всем полям, определенным в данном модуле, и удаление
  // всех экземпляров поля, их данных и самого поля.
  foreach (array_keys(_job_post_installed_fields()) as $field) {
    field_delete_field($field);
  }

  // Проход в цикле по остальным экземплярам поля, прикрепленным к типу
  // контента job_post (вроде поля тела), и поочередное их удаление.
  $instances = field_info_instances('node', 'job_post');
  foreach ($instances as $instance_name => $instance) {
    field_delete_instance($instance);
  }

  // Удаление типа контента.
  node_type_delete('job_post');

  // Удаление информации о полях.
  field_purge_batch(1000);
}

```

Резюме

В этой главе были описаны основы использования базовых возможностей Drupal 7 по созданию пользовательского типа контента с дополнительными полями, кроме заголовка и тела, созданию произвольного типа поля и программному добавлению в модуль новых полей. В следующей главе мы рассмотрим темы и применение визуальных стилей к контенту, выводимому средой Drupal на сайте.

ГЛАВА 9

Система тем

Изменение HTML или другой разметки, генерируемой Drupal, требует знания уровня, который образует систему тем. В настоящей главе вы узнаете, как работает эта система тем, и познакомитесь с некоторыми практическими рекомендациями по использованию ядра Drupal 7. И вот первая рекомендация: не следует редактировать HTML-код в файлах модулей для изменения внешнего вида и поведения сайта. Поступая так, вы просто создадите собственную систему управления контентом и лишитесь огромного преимущества — использования программной системы с открытым исходным кодом, поддерживаемым международным сообществом. Переопределяйте, но не изменяйте!

Темы

В терминологии Drupal темы — это коллекции файлов, которые задают внешний вид и поведение сайта. Можно загрузить уже готовые темы со страницы <http://drupal.org/project/themes> или создать собственную тему, и именно об этом пойдет речь в данной главе. Темы состояются в основном из элементов, хорошо знакомых веб-дизайнерам: таблицы стилей, изображения, файлы JavaScript и т.д. Отличие между темой Drupal и обычным HTML-сайтом заключается в специальных шаблонных файлах. Как правило, эти файлы содержат крупные разделы HTML и небольшие специализированные фрагменты, которые заменяются динамическим содержимым, когда Drupal генерирует страницу. Такие специальные шаблонные файлы можно создать практически для любого контейнера контента в Drupal — всей страницы, областей, блоков, нод, комментариев и даже полей. Мы немного ознакомимся с процессом создания нескольких шаблонных файлов уровня компонента, но сначала рассмотрим установку полностью готовой темы из сайта Drupal.org и ознакомимся с компонентами, образующими тему.

Установка готовой темы

Для Drupal доступны сотни тем. Если вам нужен быстрый и легкий способ создания и запуска сайта, можно просмотреть готовые темы на странице www.drupal.org/project/themes. Лишь выберите в раскрывающемся списке Filter by compatibility (Фильтр по совместимости) вариант 7.x, чтобы просматривать только темы, перенесенные в Drupal 7.

На заметку! Необходимо выбрать одну из тем, у которой есть версия для Drupal 7. Темы для Drupal 6 и предшествующих версий темы не работают в сайтах Drupal 7 из-за изменившейся внутренней структуры.

При просмотре тем вы заметите, что среди них часто встречаются темы с пометками “starter theme” (начальная тема). Начальные темы задуманы как надежный фундамент для создания новых тем. Они содержат объемную встроенную документацию, а также полезные средства и функциональность. Преимущество начальной темы в том, что она предлагает структуру, поверх которой можно добавлять графические элементы и цвета без необходимости начинать все “с чистого листа”. Темы, не отмеченные как начальные, уже содержат примененные графические эффекты — изображения, цвета, шрифты и т.д. — и их можно использовать с очень незначительными модификациями.

В целях демонстрации выполним установку темы *Picture Reloaded*. В ней нет ничего особенного, кроме адаптации для работы с Drupal 7. Зайдите на страницу этой темы (http://drupal.org/project/picture_reloaded) и скопируйте URL ссылки для версии Drupal 7. Вернитесь на свой сайт, щелкните на ссылке Appearance (Внешний вид) в верхней части страницы, и на открывшейся странице Appearance щелкните на ссылке Install new theme (Установить новую тему). На форме для загрузки новой темы вставьте загрузочный URL темы *Picture Reloaded* в текстовое поле Install from a URL (Установить с адреса), а затем щелкните на кнопке Install (Установить). Drupal загрузит и сохранит тему в каталоге `sites/all/themes` на вашем компьютере. Эту тему можно сделать темой по умолчанию — для этого понадобится снова зайти на страницу Appearance и щелкнуть на ссылке Set default (Сделать стандартной).

Установка темы с сайта [Drupal.org](http://drupal.org) выполняется быстро и без проблем. С помощью описанной процедуры можно загрузить любое количество тем и протестировать их на своем сайте, но, скорее всего, однажды вы захотите создать собственную тему. В последующих разделах будет показано, как начать с нуля и создать совершенно новую тему Drupal.

Создание темы

Имеется несколько способов создания темы, и выбор зависит от начальных условий. Предположим, ваш дизайнер уже создал для сайта HTML- и CSS-файлы. Их не очень трудно преобразовать в тему Drupal.

Ниже перечислены общие шаги по созданию новой темы Drupal.

1. Создайте или модифицируйте HTML-файл для сайта.
2. Создайте или модифицируйте CSS-файл для сайта.
3. Создайте `.info`-файл с описанием новой темы для Drupal.
4. Стандартизируйте имена файлов согласно правилам Drupal.
5. Вставьте в шаблон доступные переменные.
6. Создайте дополнительные файлы для отдельных типов нод, блоков и т.д.

Начнем создание новой темы с выбора ее имени — *Grayscale*, что означает “Оттенки серого” — и создадим в папке `sites/all/themes` новый каталог с таким же именем (`sites/all/themes/grayscale`). Теперь в этом каталоге необходимо создать `.info`-файл для новой темы. Сначала создадим файл `grayscale.info` с базовой информацией, необходимой для добавления темы в реестр тем Drupal:

```
name = Grayscale
core = 7.x
engine = phptemplate
```

После сохранения этого файла тему уже можно активировать: щелкните на ссылке Appearance (Внешний вид) в верхней части страницы и найдите на открывшейся странице тему *Grayscale*. Затем щелкните на ссылке Enable and set default (Активировать и сделать стандартной), чтобы сделать эту тему стандартной. Щелкните на кнопке Home

(Домой), чтобы зайти на домашнюю страницу сайта. И вы увидите новую тему Drupal (рис. 9.1), для создания которой оказалось достаточно написать три строки кода в файле `.info`.

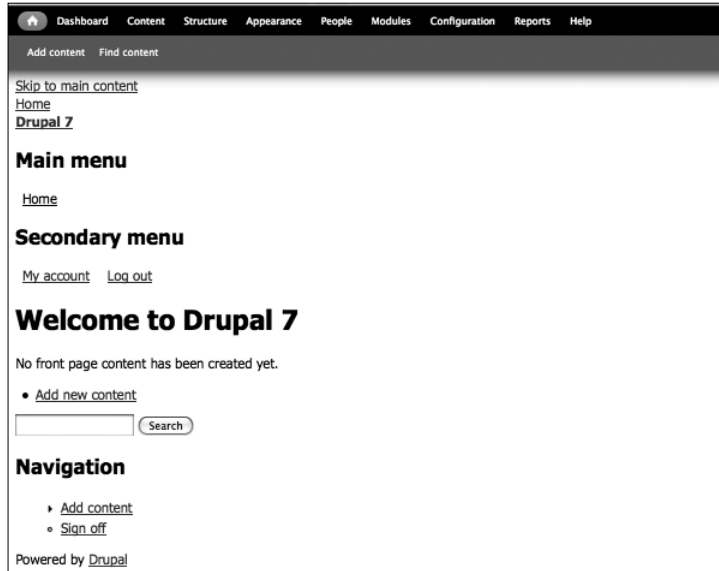


Рис. 9.1. Сайт с применением темы Grayscale

Конечно, премию за креативный дизайн вы вряд ли получите, но данный процесс демонстрирует, насколько просто создать тему Drupal с нуля. Теперь можно немного расширить сайт, применив CSS для переупорядочения и стилизации элементов. Сначала понадобится создать в каталоге темы Grayscale новый подкаталог по имени `css`. Собирать все CSS-файлы в отдельный каталог не обязательно, но это позволяет другим не блуждать по всему каталогу темы в поиске всех CSS-файлов. В каталоге `css` создайте новый файл с именем `style.css`. Имя выбирается произвольно, но во многих темах Drupal основные `.css`-файлы темы называются именно `style.css`, и это стало фактически соглашением об именовании.

Теперь в теме нужно указать, что к ней применяется файл `style.css`. Для этого в файл `grayscale.info` добавьте следующую строку:

```
stylesheets[all][] = css/style.css
```

Она означает, что ко всем носителям, применяемым для отображения сайта (экран, проектор и принтер) должна применяться таблица стилей `style.css`. Можно создавать отдельные таблицы стилей и для конкретных носителей — например, для принтера — с помощью следующей строки:

```
stylesheets[print][] = css/print.css
```

А так можно указать, что таблица стилей должна применяться и для экрана, и для проектора:

```
stylesheets[screen, projector][] = theScreenProjectorStyle.css
```

В нашем случае выбран файл для всех носителей.

Теперь рассмотрим структуру, используемую в Drupal для вывода страницы, чтобы можно было указать идентификаторы CSS и классы, к которым применяются стили.

Если вы пользуетесь браузером Firefox, то удобно загрузить и установить подключаемый модуль Firebug (<http://getfirebug.com>). Для Internet Explorer рекомендуется загрузить и установить компонент Developers Toolbar (www.microsoft.com/downloads/en/details.aspx?FamilyID=95e06cbe-4940-4218-b75d-b8856fced535), а для Safari — встроенный веб-инспектор. Все эти три инструмента позволяют просматривать структуру сайта и легко определять, какие идентификаторы CSS и классы должны применяться для стилизации. На рис. 9.2 показаны типы информации, выводимые подключаемым модулем Firebug для Firefox при просмотре страницы.



Рис. 9.2. Информация, выведенная средством Firebug для Firefox

Загрузите одно из этих средств, если у вас еще его нет, и после установки воспользуйтесь опцией Inspection (Исследование) для просмотра разметки HTML и разделов div, сгенерированных Drupal.

После этого нужно определить стилизацию для идентификаторов CSS и классов. Но перед этим давайте посмотрим на исходный код страниц нашего сайта, т.е. HTML-код, сгенерированный Drupal для домашней страницы нового сайта, и особенно структуру разделов div. Для краткости опустим HTML-код между дескрипторами div. Чтобы увидеть весь текст страницы, просто щелкните правой кнопкой мыши в окне браузера и в

появившемся контекстном меню выберите пункт View Page Source (Просмотр исходного текста страницы). Ниже приведена только структура разделов между дескрипторами `<body>` и `</body>`.

```
<body class="html front logged-in one-sidebar sidebar-first page-node
toolbar toolbar-drawer" >
  <div id="skip-link"> ... </div>
  <div class="region region-page-top"> ... </div>
  <div id="page-wrapper">
    <div id="header">
      <div class="section clearfix">
        <a id="logo" ... ></a>
        <div id="name-and-slogan"></div>
      </div>
    </div>
    <div id="navigation">
      <div class="section">
        <ul id="main-menu"> ... </ul>
        <ul id="seconary-menu"> ... </ul>
      </div>
    </div>
    <div id="main-wraper">
      <div id="main">
        <div id="content"> ... </div>
        <div id="sidebar-first" class="column sidebar"> ... </div>
      </div>
    </div>
    <div id="footer"> ... </div>
  </div>
</body>
```

Внутри дескрипторов `div` содержатся большие фрагменты кода, но в нашем упражнении важно уяснить общую структуру разделов, чтобы потом знать, как добавлять определения стилей в файл `css/style.css`. В листинге 9.1 приведены определения CSS, которые были использованы для создания графического представления на рис. 9.3.

Листинг 9.1. Содержимое файла `style.css`

```
body {
  background-color: #c6c6c6;
}

#page {
  background-color: #c6c6c6;
}

#skip-link {
  width: 960px;
  margin-right: auto;
  margin-left: auto;
  background-color: #c6c6c6;
}

#header {
  width: 960px;
  background-color: #ffffff;
```

```
margin-right: auto;
margin-left: auto;
margin-top: 10px;
height: 40px;
padding-top: 10px;
border-top: 3px solid #000;
border-bottom: 3px solid #000;
}

#logo {
  float: left;
  margin-left: 20px;
}

a#logo {
  text-decoration: none;
}

#name-and-slogan {
  float: left;
  margin-left: 100px;
}

#site-name a {
  text-decoration: none;
}

#navigation {
  width: 960px;
  margin-right: auto;
  margin-left: auto;
  background-color: #c6c6c6;
  height: 45px;
}

#navigation h2 {
  display: none;
}

ul#main-menu {
  background-color: #EEE;
  height: 25px;
}

ul#main-menu {
  text-decoration: none;
  padding-top: 5px;
}

ul#main-menu li a {
  text-decoration: none;
  padding-right: 10px;
}

ul#secondary-menu {
  background-color: #333;
```

```

    height: 25px;
}

ul#secondary-menu li a {
    text-decoration: none;
    color: #fff;
    padding-right: 10px;
    height: 25px;
    border-right: 1px solid #fff;
}

ul#secondary-menu a:hover {
    color: #ff0000;
}

#main-wrapper {
    clear: both;
    background-color: #ffffff;
    width: 960px;
    margin-right: auto;
    margin-left: auto;
}

#main {
    width: 960px;
    margin: 5px auto;
}

#content {
    width: 775px;
    float: right;
    padding-left: 15px;
}

#sidebar-first {
    float: left;
    width: 130px;
    margin: 0;
    padding: 20px;
    background-color: #EEE;
}

#footer {
    width: 920px;
    padding: 20px;
    margin-right: auto;
    margin-left: auto;
    clear: both;
    min-height: 100px;
    background-color: #333;
    color: #fff;
}

#footer a {
    color: #fff;
}

```

После сохранения файла `style.css` вернитесь на домашнюю страницу сайта. Она должна выглядеть примерно так, как на рис. 9.3.

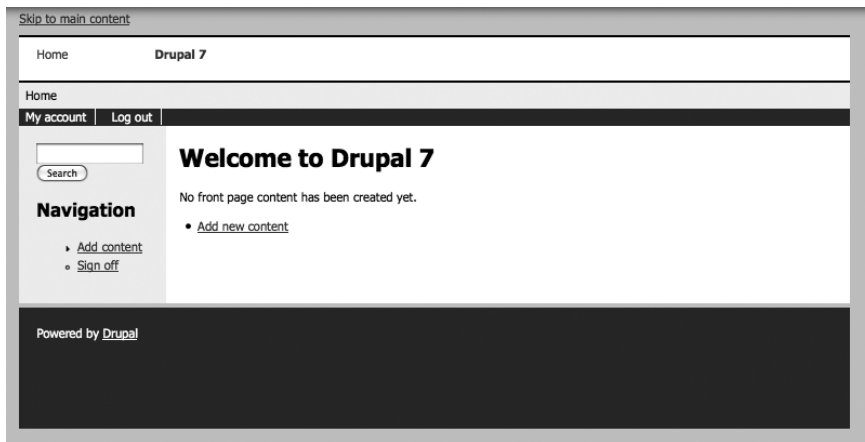


Рис. 9.3. Сайт после применения дополнений в стилевой таблице

Итак, с помощью файла `.info`, содержащего всего четыре строки, и нескольких строк CSS мы смогли создать совершенно новую тему Drupal. При этом не пришлось создавать шаблонные файлы, HTML-разметку или писать PHP-код — все заменила простая и мощная темизация Drupal.

Одной из причин такой легкости создания темы Grayscale было то, что Drupal поставляется с предопределенным набором шаблонных файлов, которые применяются к теме, а в самом дистрибутиве темы этих файлов нет. В следующем разделе мы рассмотрим различные шаблонные файлы.

Файл `.info`

Файл `.info` для темы Grayscale содержит минимальный объем информации, чтобы зарегистрировать тему и сделать ее доступной для выбора на странице Appearance (Внешний вид). В большинстве случаев в состав темы нужно добавлять собственные области, дополнительные стилевые таблицы и файлы JavaScript. Сейчас мы рассмотрим, как можно расширить файл `.info`, добавляя в него каждый из этих атрибутов.

Добавление областей в тему

Область — это, в сущности, раздел страницы на сайте. При создании темы Grayscale мы использовали стандартные области, которые Drupal создает автоматически: `sidebar first` (первая врезка), `sidebar second` (вторая врезка), `content` (содержимое), `header` (заголовок), `footer` (концовка), `highlighted` (выделенный), `help` (подсказка), `page_bottom` (низ страницы) и `page_top` (верх страницы). Но часто бывает, что тему необходимо разбить на дополнительные области, и это можно сделать с помощью сочетания описания области в файле `.info` и вставки этой области в файл `page.tpl.php`.

Синтаксис определения новой области в теме имеет следующий вид:

```
regions[alerts] = Alerts // Оповещения
regions[featured] = Featured Articles // Имеющиеся статьи
regions[socialnetworks] = Social Networks // Социальные сети
```


В файле `.info` можно определить сколько угодно областей, но в любом случае он должен содержать области `page_bottom`, `page_top`, `help` и `content` — они необходимы для корректной работы ядра. После этого необходимо поместить ссылки на новые области в файле `page.tpl.php`. Процесс вывода области на странице выглядит так:

```
<div id="alerts">
<?php print render($page['alerts']); ?>
</div> <!-- /alerts -->
```

Добавление CSS-файлов в тему

При создании темы Grayscale мы создали единственный CSS-файл, который содержал все стили, необходимые для достижения наших целей. Но бывает так, что нужно более одной стилиевой таблицы, или понадобится стилиевая таблица для нестандартного устройства, на котором будет просматриваться сайт. Все это выполняется с помощью добавления стилиевых таблиц с использованием следующего синтаксиса (если все стилиевые таблицы находятся в подкаталоге `css` каталога тем):

```
// добавление стилиевой таблицы для работы с цветами для всех носителей
stylesheets[all][] = css/colors.css

// добавление стилиевой таблицы только для печати
stylesheets[print][] = css/print.css

// добавление стилиевой таблицы только для воспроизведения через проектор
stylesheets[projector][] = css/showtime.css

// добавление стилиевой таблицы для экрана
stylesheets[screen][] = css/style.css

// добавление стилиевой таблицы для экрана и проектора
stylesheets[screen, projector] = css/showit.css
```

Добавление файлов JavaScript

Если в теме используется JavaScript, то код JavaScript рекомендуется создавать и хранить во внешних файлах. Для включения этих файлов в тему необходимо перечислить их все в JavaScript-коде в файле `.info`. Если, к примеру, все файлы JavaScript находятся в подкаталоге темы с именем `js`, то эти файлы включаются с помощью следующего синтаксиса:

```
scripts[] = js/jcarousel.js
```

Добавление параметров в тему

Иногда требуется сделать тему настраиваемой без применения шаблонных файлов или CSS. Например, может понадобиться добавить возможность для администратора сайта, чтобы он мог изменить стандартный шрифт и стандартный размер шрифта. Это можно сделать с помощью параметров. Необходимо вставить в файл `.info` определения параметров:

```
settings[font_family] = 'ff-sss'
settings[font_size] = 'fs-12'
```

Добавьте в файл `grayscale.info` эти определения и переходите к выполнению следующих шагов.

А следующим шагом будет обеспечение администратора сайта средствами для изменения этих значений. Для этого создадим в каталоге темы Grayscale файл `theme-settings.php` и добавим элементы формы, необходимые для приема сведений

о семействе шрифтов и размере шрифта. В файле `theme-settings.php` будет задействована функция `hook_form_system_theme_settings_alter()`, которая добавляет поля для ввода этих значений:

```
<?php
function grayscale_form_system_theme_settings_alter(&$form, &$form_state)
{
    $form['styles'] = array(
        '#type' => 'fieldset',
        '#title' => t('Style settings'), // Параметры стиля
        '#collapsible' => FALSE,
        '#collapsed' => FALSE,
    );
    $form['styles']['font'] = array(
        '#type' => 'fieldset',
        '#title' => t('Font settings'), // Параметры шрифта
        '#collapsible' => TRUE,
        '#collapsed' => TRUE,
    );
    $form['styles']['font']['font_family'] = array(
        '#type' => 'select',
        '#title' => t('Font family'), // Семейство шрифтов
        '#default_value' => theme_get_setting('font_family'),
        '#options' => array(
            'ff-sss' => t('Helvetica Neue, Trebuchet MS, Arial, Nimbus Sans L,
FreeSans, sans-serif'),
            'ff-ssl' => t('Verdana, Geneva, Arial, Helvetica, sans-serif'),
            'ff-a' => t('Arial, Helvetica, sans-serif'),
            'ff-ss' => t('Garamond, Perpetua, Nimbus Roman No9 L, Times New
Roman, serif'),
            'ff-sl' => t('Baskerville, Georgia, Palatino, Palatino Linotype,
Book Antiqua, URW Palladio L, serif'),
            'ff-m' => t('Myriad Pro, Myriad, Arial, Helvetica, sans-serif'),
            'ff-l' => t('Lucida Sans, Lucida Grande, Lucida Sans Unicode,
Verdana, Geneva, sans-serif'),
        ),
    );
    $form['styles']['font']['font_size'] = array(
        '#type' => 'select',
        '#title' => t('Font size'), // Размер шрифта
        '#default_value' => theme_get_setting('font_size'),
        '#description' => t('Font sizes are always set in relative units - the
sizes shown are the pixel value equivalent.'),
        // Размеры шрифтов всегда указываются в относительных единицах -
        // размеры, приведенные здесь, эквиваленты размерам в пикселях.
        '#options' => array(
            'fs-10' => t('10px'),
            'fs-11' => t('11px'),
            'fs-12' => t('12px'),
            'fs-13' => t('13px'),
            'fs-14' => t('14px'),
            'fs-15' => t('15px'),
            'fs-16' => t('16px'),
        ),
    );
}
```

После сохранения этого файла перейдите на страницу Appearance (Внешний вид) и щелкните на ссылке Settings (Параметры) для темы Grayscale. Теперь в нижней части формы должны появиться только что добавленные поля для задания параметров стиля. Щелкните на ссылке Font Settings (Параметры шрифта), чтобы раскрыть форму, показанную на рис. 9.4.

Рис. 9.4. Параметры настройки шрифта

И последний шаг — применение значений, указанных администратором сайта, в теме. Мы сделаем это, добавив параметры для семейства шрифтов и размера шрифта в переменную `$classes` нашей темы. Для этого понадобится создать файл `template.php`. Данный файл используется для различных действий над темами. Подробно с его содержанием мы ознакомимся далее в главе, а пока создадим файл `template.php` в каталоге темы Grayscale и функцию `hook_process_HOOK()` для добавления значений параметров в переменную `$classes`. В качестве имени функции `hook_process_HOOK()` возьмем `grayscale_process_html()`, где `grayscale` — имя темы, а `html` — имя файла `.tpl.php`, который необходимо переопределить. С помощью этой же функции `hook_process_HOOK()` можно переопределить и любой другой файл темы.

```
<?php
/**
 * Переопределение или вставка переменных в шаблоне html.
 */
function grayscale_process_html(&$vars) {
    // Добавление классов для стилей шрифтов.
    $classes = explode(' ', $vars['classes']);
    $classes[] = theme_get_setting('font_family');
    $classes[] = theme_get_setting('font_size');
    $vars['classes'] = trim(implode(' ', $classes));
}
```

После установки переменных они будут применены в файле `html.tpl.php` и задействованы в дескрипторе `body` через переменную `$classes`. Файл `html.tpl.php` находится в каталоге `modules/system` и относится к ядру. Ниже в этой главе будет показано, как с помощью файла `html.tpl.php` переопределять шаблоны ядра.

```
<body class="<?php print $classes; ?>" <?php print $attributes;?>
```

Если вывести содержимое переменной `$classes`, то вы увидите следующие значения (при условии, что они не изменены где-то еще):

```
html front logged-in one-sidebar sidebar-first page-node toolbar
toolbar-drawer ff-sss fs-12
```

Значения `ff-sss` и `fs-12` добавлены в конец содержимого переменной `$classes`. Теперь осталось только создать CSS-код для применения каждого из этих параметров. Добавим в файл `css/style.css` стили для каждого варианта, определенного в созданном ранее файле `theme-settings.php`:

```
/* Семейство шрифтов */
.ff-sss {font-family: "Helvetica Neue", "Trebuchet MS", Arial,
"Nimbus Sans L", FreeSans, sans-serif;}
.ff-ssl {font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;}
.ff-a {font-family: Arial, Helvetica, sans-serif;}
.ff-ss {font-family: Garamond, Perpetua, "Nimbus Roman No9 L",
"Times New Roman", serif;}
.ff-sl {font-family: Baskerville, Georgia, Palatino, "Palatino Linotype",
"Book Antiqua", "URW Palladio L", serif;}
.ff-m {font-family: "Myriad Pro", Myriad, Arial, Helvetica, sans-serif;}
.ff-l {font-family: "Lucida Sans", "Lucida Grande", "Lucida Sans Unicode",
Verdana, Geneva, sans-serif;}

/* Базовые размеры шрифта */
.fs-10 {font-size:0.833em}
.fs-11 {font-size:0.917em}
.fs-12 {font-size:1em}
.fs-13 {font-size:1.083em}
.fs-14 {font-size:1.167em}
.fs-15 {font-size:1.25em}
.fs-16 {font-size:1.333em}
```

Теперь попробуйте изменить семейство шрифтов и размер шрифта: зайдите на страницу Appearance (Внешний вид) и щелкните на ссылке Settings (Параметры) для темы Grayscale.

Шаблонные файлы

У некоторых тем имеется много различных шаблонных файлов, а у других — как у нашей темы Grayscale — только файлы `.info` и `.css`. Количество шаблонных файлов, которые нужны для темы, зависит от того, насколько настраиваемыми вы хотите сделать стандартные шаблоны Drupal для различных компонентов своего сайта. При просмотре различных шаблонных файлов не забывайте, что если ваша тема не предоставляет один из описанных здесь шаблонных файлов, Drupal самостоятельно применит шаблонные файлы из ядра Drupal для каждого компонента — страницы, ноды, комментария, поля. В последующих разделах вы узнаете, где находятся все эти файлы.

Крупный план

С каждой темой Drupal связано несколько файлов `tpl.php`. У некоторых тем имеется лишь базовый файл `html.tpl.php`, который можно считать файлом темы и который содержит все, что размещено над дескриптором `<body>` в обычном веб-сайте на основе HTML, а также файл `page.tpl.php`, представляющий собой все между дескрипторами `<body>` и `</body>` в обычном веб-сайте на основе HTML. В некоторых темах задействуются возможности Drupal по темизации отдельных компонентов страниц с помощью дополнительных файлов тем:

- `node.tpl.php` — определяет способ визуализации нод на странице;
- `field.tpl.php` — определяет способ визуализации полей на странице;
- `block.tpl.php` — определяет способ визуализации блоков на странице.

Прародителем всех шаблонных файлов является `html.tpl.php`, который задает общую компоновку сайта. Другие шаблонные файлы содержатся внутри `page.tpl.php`, как показано на рис. 9.5.

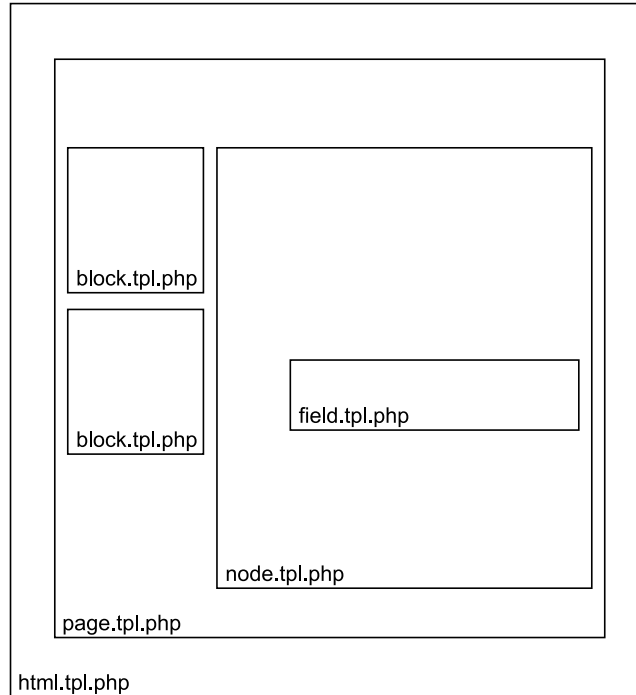


Рис. 9.5. Файл `page.tpl.php` содержит остальные шаблоны

Вставка файлов `html.tpl.php`, `page.tpl.php`, `block.tpl.php`, `node.tpl.php` и `field.tpl.php` на рис. 9.5 выполняется автоматически системой тем во время построения страницы. Если какая-то тема не содержит один из этих файлов или все, Drupal использует стандартные шаблоны из ядра Drupal (табл. 9.1).

Таблица 9.1. Шаблонные файлы из ядра Drupal

Шаблонный файл	Местоположение	Назначение
<code>html.tpl.php</code>	<code>modules/system</code>	Главный шаблонный файл сайта, который содержит все элементы из раздела <code><head>...</head></code> HTML-страницы сайта
<code>page.tpl.php</code>	<code>modules/system</code>	Определяет все содержимое страницы между дескрипторами <code><body></code> и <code></body></code> включительно; при работе с общей структурой основной компоновки сайта изменять необходимо этот файл
<code>region.tpl.php</code>	<code>modules/system</code>	Определяет расположение и отображение областей в сайте
<code>node.tpl.php</code>	<code>modules/node</code>	Определяет расположение и отображение нод в сайте
<code>block.tpl.php</code>	<code>modules/block</code>	Определяет расположение и отображение блоков в сайте
<code>field.tpl.php</code>	<code>modules/field/</code> <code>theme</code>	Определяет расположение и отображение полей в сайте

Прежде чем приступить к созданию собственных вариантов некоторых из этих шаблонных файлов для нашей темы Grayscale, давайте кратко ознакомимся со структурой и содержимым перечисленных шаблонных файлов из ядра Drupal.

Файл `html.tpl.php`

Это стандартный шаблон, который выводит базовую структуру HTML-разметки страницы на сайте Drupal. Он ориентирован на элементы между открывающим дескриптором `<HTML>` и начальным дескриптором `<body>`. В приведенном ниже коде можно увидеть, что файл `html.tpl.php` содержит определение `DOCTYPE`, определения RDF (Resource Description Framework — инфраструктура описания ресурсов), HTML, несколько дескрипторов `DIV` и фрагменты PHP-кода для вывода контента, связанного с различными переменными, наподобие приведенных в табл. 9.2. Этот шаблонный файл относительно прост, но он демонстрирует мощь темы Drupal и возможность отображения динамического контента за счет установки переменных во время выполнения и замены механизмом обработки тем этих значений конкретным контентом. Значения переменных можно задать разными способами: через параметры в URL, в зависимости от того, является ли пользователь анонимным или зарегистрированным, от роли пользователя, если он вошел на сайт, и других контекстов, которые позволяют определить, что следует вывести на странице. В других шаблонных файлах можно встретить и другие примеры этого, но важно понять значение небольших фрагментов PHP-кода, которые находятся в файле `html.tpl.php`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php print
$language->language; ?>"
  version="XHTML+RDFa 1.0" dir="<?php print $language->dir; ?>"
  <?php print $rdf_namespaces;
?>>

<head profile="<?php print $grddl_profile; ?>">
  <?php print $head; ?>
  <title><?php print $head_title; ?></title>
  <?php print $styles; ?>
  <?php print $scripts; ?>
</head>

<body class="<?php print $classes; ?>" <?php print $attributes;?>>
  <div id="skip-link">
    <a href="#main-content"><?php print t('Skip to main content'); ?></a>
  </div>
  <?php print $page_top; ?>
  <?php print $page; ?>
  <?php print $page_bottom; ?>
</body>
</html>
```

В табл. 9.2 описаны все переменные, которые сделаны доступными в файле `html.tpl.php` через различные функции процессора и препроцессора шаблонов. Например, переменная `$css` содержит список CSS-файлов для текущей страницы. Эти CSS-файлы определены в `.info-файле` темы с помощью значений `style sheets[all][ ]`, объявленных в данном файле.

На заметку! Список переменных для каждого шаблонного файла представляет переменные, доступные для шаблона. Не обязательно задействовать все переменные шаблонного файла, достаточно использовать лишь те, которые нужны для поддержки функциональных и технических требований — за исключением переменных, применяемых в шаблоне `html.tpl.php`.

Таблица 9.2. Переменные, доступные для использования в файле `html.tpl.php`

Переменная	Что содержит
<code>\$css</code>	Массив CSS-файлов для текущей страницы
<code>\$language</code> (объект)	Язык, на котором отображается сайт
<code>\$language->language</code>	Текстовое представление
<code>\$language->dir</code>	Направление вывода символов: <code>ltr</code> — слева направо или <code>rtl</code> — справа налево
<code>\$rdf_namespaces</code>	Все префиксы пространства имен RDF, используемые в HTML-документе
<code>\$grddl_profile</code>	Профиль GRDDL, который позволяет агентам извлекать данные RDF
<code>\$head_title</code>	Вариант заголовка страницы для дескриптора <code>title</code>
<code>\$head</code>	Разметка для раздела <code>HEAD</code> : дескрипторы <code>meta</code> , <code>keyword</code> и т.д.
<code>\$styles</code>	Стилевые дескрипторы для импорта всех CSS-файлов для данной страницы
<code>\$scripts</code>	Стилевые дескрипторы для загрузки JavaScript-файлов и параметров для данной страницы
<code>\$page_top</code>	Первоначальная разметка из всех модулей, которые изменили данную страницу; эта переменная должна быть выведена перед остальным динамическим контентом
<code>\$page</code>	Готовый для отображения контент страницы; Drupal заменяет <code>\$page</code> содержимое <code>\$page</code> содержимым <code>page.tpl.php</code> (см. следующий раздел)
<code>\$page_bottom</code>	Последняя завершающая разметка из всех модулей, которые изменили данную страницу; эта переменная должна быть выведена после остального динамического контента
<code>\$classes</code>	Строка классов для контекстуальной стилизации с помощью CSS; по умолчанию это <code>html front logged-in one-sidebar sidebar-first page-node toolbar toolbar-drawer</code> — каждый из этих классов можно использовать в качестве суффиксов для идентификаторов <code>DIV</code> и классов

Файл `page.tpl.php`

Следующим мы рассмотрим шаблонный файл `page.tpl.php`. Он ориентирован на элементы, которые находятся между дескрипторами `<body>` и `</body>`, и содержит HTML-структуру страницы, в том числе дескрипторы `div` и фрагменты PHP-кода. Просмотрите еще раз шаблон `html.tpl.php` — в нем присутствует фрагмент `<?php print $page; ?>`, где значением `$page` как раз и является содержимое `page.tpl.php`.

Этот шаблонный файл определяет структуру страницы, как она выводится для пользователя. В примере в самом начале этой главы все элементы, для которых выполнялась стилизация в теме Grayscale — это элементы, которые отображаются с помощью шаблона `page.tpl.php`. В приведенном ниже коде вы увидите все элементы, которые визуализируются на странице, и условную логику, которая определяет, что выводится на странице, и как страница структурирована. Подобно `html.php.tpl`, файл `page.tpl.php` содержит смесь фрагментов HTML и PHP, при этом в PHP-фрагментах предусмотрена условная логика для определения, задано ли какое-то значение, и отображения значений некоторых переменных, которые могут содержать что угодно, от просто числа вроде 42 до сложной HTML-структуры, включающей JavaScript. Примером может служить первая переменная из файла `page.tpl.php` — с именем `$logo`. Она содержит HTML-разметку, необходимую для отображения логотипа сайта, и поэтому должна быть предусмотрена условная логика для проверки, задан ли этот логотип.

Структура файла `page.tpl.php`, который находится в каталоге `/modules/system`, выглядит следующим образом:

```
<div id="page-wrapper"><div id="page">

  <div id="header"><div class="section clearfix">

    <?php if ($logo): ?>
      <a href="<?php print $front_page; ?>" title="<?php print t('Home'); ?>"
        rel="home" id="logo">
        " />
      </a>
    <?php endif; ?>

    <?php if ($site_name || $site_slogan): ?>
      <div id="name-and-slogan">
        <?php if ($site_name): ?>
          <?php if ($title): ?>
            <div id="site-name"><strong>
              <a href="<?php print $front_page; ?>"
                title="<?php print t('Home'); ?>" rel="home">
                <span><?php print $site_name; ?></span>
              </a>
            </strong></div>
          <?php else: /* При пустом заголовке контента используется h1 */ ?>
            <h1 id="site-name">
              <a href="<?php print $front_page; ?>"
                title="<?php print t('Home'); ?>" rel="home">
                <span><?php print $site_name; ?></span>
              </a>
            </h1>
          <?php endif; ?>
        <?php endif; ?>

        <?php if ($site_slogan): ?>
          <div id="site-slogan"><?php print $site_slogan; ?></div>
        <?php endif; ?>
      </div> <!-- /#name-and-slogan -->
    <?php endif; ?>
    <?php print render($page['header']); ?>

  </div></div> <!-- /.section, /#header -->
```



```

<?php if ($main_menu || $secondary_menu): ?>
  <div id="navigation"><div class="section">
    <?php print theme('links__system_main_menu', array('links' => $main_menu,
'attributes' => array('id' => 'main-menu', 'class' => array('links',
'clearfix')), 'heading' => t('Main menu'))); ?>
    <?php print theme('links__system_secondary_menu',
array('links' => $secondary_menu,
'attributes' => array('id' => 'secondary-menu',
'class' => array('links', 'clearfix')),
'heading' => t('Secondary menu'))); ?>
  </div></div> <!-- /.section, /#navigation -->
<?php endif; ?>

<?php if ($breadcrumb): ?>
  <div id="breadcrumb"><?php print $breadcrumb; ?></div>
<?php endif; ?>

<?php print $messages; ?>

<div id="main-wrapper"><div id="main" class="clearfix">

  <div id="content" class="column"><div class="section">
    <?php if ($page['highlighted']): ?><div id="highlighted">
      <?php print render($page['highlighted']); ?>
    </div><?php endif; ?>
    <a id="main-content"></a>
    <?php print render($title_prefix); ?>
    <?php if ($title): ?><h1 class="title" id="page-title">
      <?php print $title; ?></h1>
    <?php endif; ?>
    <?php print render($title_suffix); ?>
    <?php if ($tabs): ?><div class="tabs">
      <?php print render($tabs); ?>
    </div><?php endif; ?>
    <?php print render($page['help']); ?>
    <?php if ($action_links): ?><ul class="action-links">
      <?php print render($action_links); ?>
    </ul><?php endif; ?>
    <?php print render($page['content']); ?>
    <?php print $feed_icons; ?>
  </div></div> <!-- /.section, /#content -->

  <?php if ($page['sidebar_first']): ?>
    <div id="sidebar-first" class="column sidebar"><div class="section">
      <?php print render($page['sidebar_first']); ?>
    </div></div> <!-- /.section, /#sidebar-first -->
  <?php endif; ?>

  <?php if ($page['sidebar_second']): ?>
    <div id="sidebar-second" class="column sidebar"><div class="section">
      <?php print render($page['sidebar_second']); ?>
    </div></div> <!-- /.section, /#sidebar-second -->
  <?php endif; ?>
</div></div> <!-- /#main, /#main-wrapper -->
<div id="footer"><div class="section">
  <?php print render($page['footer']); ?>
</div></div> <!-- /.section, /#footer -->

</div></div> <!-- /#page, /#page-wrapper -->

```

Стандартные переменные, доступные в файле `page.tpl.php`, перечислены в табл. 9.3.

Таблица 9.3. Стандартные переменные, доступные в файле `page.tpl.php`

Переменная	Содержимое
<code>\$base_path</code>	Базовый URL пути установки Drupal; по умолчанию содержит /
<code>\$directory</code>	Каталог, в котором находится шаблон, например, <code>modules/system</code> или <code>themes/bartik</code>
<code>\$is_front</code>	TRUE, если данная страница является передней страницей
<code>\$logged_in</code>	TRUE, если пользователь зарегистрирован и вошел на сайт
<code>\$is_admin</code>	TRUE, если у пользователя есть право доступа к страницам администрирования
<code>\$front_page</code>	URL передней страницы; при обращении к передней странице используйте этот адрес. Он содержит языковой домен или префикс
<code>\$logo</code>	Путь к изображению логотипа, определенный в конфигурации темы
<code>\$site_name</code>	Имя сайта; пусто, если отключено в параметрах темы
<code>\$site_slogan</code>	Слоган сайта; пусто, если отключен в параметрах темы
<code>\$main_menu</code> (массив)	Массив со ссылками на главное меню сайта, если оно сконфигурировано
<code>\$secondary_menu</code> (массив)	Массив со ссылками на вторичное меню сайта, если оно сконфигурировано
<code>\$breadcrumb</code>	Навигационная цепочка данной страницы
<code>\$title_prefix</code>	Массив, заполняемый модулями; содержит дополнительные выходные данные для вывода перед главным дескриптором <code>title</code> , который присутствует в шаблоне
<code>\$title</code>	Заголовок страницы, для использования в реальном HTML-контенте
<code>\$title_suffix</code> (массив)	Массив, заполняемый модулями; содержит дополнительные выходные данные для вывода после главного дескриптора <code>title</code> , который присутствует в шаблоне
<code>\$message</code>	Сообщения о состоянии и ошибках, которые должны быть хорошо заметны
<code>\$tabs</code> (массив)	Ссылки для закладок на все подстраницы данной страницы
<code>\$action_links</code> (массив)	Действия, локальные на странице, наподобие <code>Add menu</code> (Добавить меню) в интерфейсе администрирования меню
<code>\$feed_icons</code>	Строка со всеми значками для ленты новостей для данной страницы
<code>\$node</code>	Объект ноды, если имеется автоматически загружаемая нода, связанная со страницей, и идентификатор ноды — второй аргумент в пути страницы (например, <code>node/12345</code> и <code>node/12345/revisions</code> , но не <code>comment/reply/12345</code>)

Переменные `$page['help']`, `$page['highlighted']`, `$page['content']`, `$page['sidebar_first']`, `$page['sidebar_second']`, `$page['header']` и `$page['footer']` представляют области на странице. Область представляет собой физические контейнеры, которые администратор сайта может назначить любому эле-

менту уровня блока (например, форме входа, блоку поиска, ноде, представлению или меню). Если в `.info`-файле темы не указаны никакие области, то по умолчанию будут доступны только что перечисленные области. Как создавать дополнительные области, вы увидите в последующих разделах, где описано создание `.info`-файлов темы.

Файл `region.tpl.php`

Этот файл задает отображение областей на сайте. Стандартный файл `region.tpl.php` весьма прост — по сути, это просто вывод контента, назначенного области:

```
<?php if ($content): ?>
  <div class="<?php print $classes; ?>">
    <?php print $content; ?>
  </div>
<?php endif; ?>
```

Переменные, доступные этому шаблонному файлу по умолчанию, перечислены в табл. 9.4.

Таблица 9.4. Стандартные переменные, доступные в файле `region.tpl.php`

Переменная	Содержимое
<code>\$content</code>	Содержимое области — обычно блоки
<code>\$classes</code>	Строка классов, которые доступны для контекстуальной стилизации с помощью CSS; их можно изменять через массив <code>\$classes</code> с помощью функций препроцессора. Стандартные значения — один или несколько следующих вариантов: <code>region</code>: текущий тип шаблона — например, "theming hook"; <code>region-[имя]</code>: имя области, в котором подчеркивания заменены дефисами. Например, для области <code>page_top</code> следует указать класс <code>region-page-top</code>.
<code>\$region</code>	Имя переменной области, которая определена в файле <code>.info</code>
<code>\$classes_array</code>	Массив значений атрибутов для класса HTML. В переменной <code>\$classes</code> он представлен в виде единой строки
<code>\$is_admin</code>	TRUE, если текущий пользователь является администратором
<code>\$is_front</code>	TRUE, если область находится на передней странице
<code>\$logged_in</code>	TRUE, если пользователь зарегистрирован и вошел на сайт

Файл `node.tpl.php`

Этот шаблонный файл задает отображение отдельных нод на сайте. Стандартный файл `node.tpl.php` находится в каталоге `modules/node` сайта. В приведенном ниже листинге содержатся элементы, похожие на другие, уже знакомые нам, шаблонные файлы — в основном это фрагменты HTML и PHP с условной логикой, которые выводят значения, присвоенные переменным. В рассматриваемом файле выводится также значение, присвоенное переменной `$content` — в данном случае это отдельная нода. В нем применяется функция сокрытия `hide()` для удаления двух элементов, которые обычно выводятся на странице при отображении ноды — комментариев и ссылок, связанных с нодой. Это осуществляется с помощью вызовов `hide($content['comments'])` и `hide($content['links'])`.

Дело в том, что обычно требуется управлять расположением комментариев и ссылок при выводе ноды, причем именно скрывая, а затем отображая их (с помощью

`print render($content['comments'])` и `print render($content['links'])`). Так, с помощью функции `hide()` можно скрыть любой обычно видимый элемент а затем позже вывести его посредством `print render()`.

Вот стандартное содержимое файла `node.tpl.php`:

```
<div id="node-<?php print $node->nid; ?>" class="<?php print $classes; ?>
clearfix"
  <?php print $attributes; ?>>

  <?php print $user_picture; ?>

  <?php print render($title_prefix); ?>
  <?php if (!$page): ?>
    <h2<?php print $title_attributes; ?>>
      <a href="<?php print $node_url; ?>"><?php print $title; ?></a>
    </h2>
  <?php endif; ?>
  <?php print render($title_suffix); ?>

  <?php if ($display_submitted): ?>
    <div class="submitted">
      <?php
        print t('Submitted by !username on !datetime',
          array('!username' => $name, '!datetime' => $date));
      ?>
    </div>
  <?php endif; ?>

  <div class="content"<?php print $content_attributes; ?>>
    <?php
      // Скрытие комментариев и ссылок для визуализации в будущем.
      hide($content['comments']);
      hide($content['links']);
      print render($content);
    ?>
  </div>

  <?php print render($content['links']); ?>

  <?php print render($content['comments']); ?>

</div>
```

Переменные, доступные по умолчанию в файле `node.tpl.php`, перечислены в табл. 9.5.

Таблица 9.5. Стандартные переменные, доступные в файле `node.tpl.php`

Переменная	Содержимое
<code>\$title</code>	Очищенный вариант заголовка
<code>\$content</code> (массив)	Массив элементов, которые составляют отображаемую ноду. Чтобы отобразить всю ноду, используется <code>render(\$content)</code> , а для вывода отдельных элементов из объекта ноды применяются описанные выше функции <code>hide()</code> и <code>show()</code>
<code>\$user_picture</code>	Изображение автора ноды из файла <code>user-picture.tpl.php</code>

Переменная	Содержимое
<code>\$date</code>	Форматированная дата создания; функции предобработки могут переформатировать ее с помощью вызова <code>format_date()</code> с переменной <code>\$created</code>
<code>\$name</code>	Пользовательское имя автора ноды с применением темы, взятое из <code>theme_username()</code>
<code>\$node_url</code>	Прямой URL данной ноды
<code>\$display_submitted</code>	Флаг (TRUE или FALSE), указывающий, нужно ли выводить отправленную информацию
<code>\$classes</code>	Строка классов, которые позволяют выполнить контекстуальную стилизацию с помощью CSS; ее можно изменять через переменную <code>\$classes_array</code> из функций препроцессора. Ниже перечислены стандартные значения: • <code>node</code> — текущий тип шаблона, например, <code>theming hook</code> (хук темизации); • <code>node-[тип]</code> — текущий тип ноды. Например, если нода является статьей блога, получится параметр <code>node-blog</code>. Имя компьютера часто приводится в краткой форме метки, читабельной для человека; • <code>node-teaser</code> — ноды в форме тизеров; • <code>node-preview</code> — ноды в режиме предварительного просмотра. • Следующие значения управляются с помощью параметров публикации нод: • <code>node-promoted</code> — ноды, выдвинутые на переднюю страницу; • <code>node-sticky</code> — ноды, упорядоченные над другими не фиксированными нодами в списках тизеров; • <code>node-unpublished</code> — неопубликованные ноды, которые видны только администраторам.
<code>\$title_prefix</code> (массив)	Массив с дополнительными выходными данными модулей, которые нужны для вывода перед дескриптором главного заголовка в шаблоне
<code>\$title_suffix</code> (массив)	Массив с дополнительными выходными данными модулей, которые нужны для вывода сразу после дескриптора главного заголовка в шаблоне
<code>\$node</code>	Полный объект ноды
<code>\$type</code>	Тип ноды: текст, страница, блог и т.д.
<code>\$comment_count</code>	Количество комментариев, присоединенных к ноде
<code>\$uid</code>	UID автора ноды
<code>\$created</code>	Время публикации ноды в формате метки времени Unix
<code>\$classes_array</code>	Массив значений атрибутов классов HTML; он же находится в виде строки в переменной <code>\$classes</code>
<code>\$zebra</code>	Возвращает либо <code>even</code> (четное), либо <code>odd</code> (нечетное) и используется для чересполосного вывода в списке тизеров
<code>\$id</code>	Позиция в ноде; увеличивается на 1 при каждом выводе
<code>\$view_mode</code>	Режим просмотра: <code>full</code> (полный) или <code>teaser</code> (тизер)

Переменная	Содержимое
\$page	Флаг состояния полной страницы (TRUE или FALSE)
\$promote	Флаг выдвижения на переднюю страницу (TRUE или FALSE)
\$sticky	Флаги для параметра фиксированного размещения
\$status	Флаг состояния опубликования
\$comment	Состояние параметров комментирования для ноды
\$readmore	Флаг, равный TRUE, если контент тизера ноды не может вместить контент тела ноды
\$is_front	Флаг, равный TRUE, если контент выводится на передней странице сайта
\$logged_in	Флаг, равный TRUE, если данный пользователь является вошедшим зарегистрированным пользователем
\$is_admin	Флаг, равный TRUE, если данный пользователь является администратором

Файл field.tpl.php

Этот шаблонный файл используется для полей тем и, в отличие от предыдущих шаблонов, не вызывается автоматически при отображении полей. Если понадобится применить этот шаблон, необходимо скопировать его из каталога `/modules/fields/templates` в каталог темы.

```
<div class="<?php print $classes; ?> clearfix"<?php print $attributes; ?>>
  <?php if (!$label_hidden) : ?>
    <div class="field-label"<?php print $title_attributes; ?>>
      <?php print $label ?>:&nbsp;</div>
    <?php endif; ?>
    <div class="field-items"<?php print $content_attributes; ?>>
      <?php foreach ($items as $delta => $item) : ?>
        <div class="field-item <?php print $delta % 2 ? 'odd' : 'even'; ?>"
          <?php print $item_attributes[$delta]; ?>>
          <?php print render($item); ?></div>
        <?php endforeach; ?>
      </div>
    </div>
```

Переменные, доступные по умолчанию в файле `field.tpl.php`, перечислены в табл. 9.6.

Таблица 9.6. Стандартные переменные, доступные в файле `field.tpl.php`

Переменная	Содержимое
\$items	Массив значений полей; для их вывода используется <code>render()</code>
\$label	Метка элемента
\$label_hidden	Флаг (TRUE или FALSE), который можно использовать для указания, нужно ли выводить метку

Переменная	Содержимое
<code>\$classes</code>	Строка классов, которые позволяют выполнить контекстуальную стилизацию с помощью CSS; ее можно изменять через переменную <code>\$classes_array</code> из функций препроцессора. Стандартные значения — одно или несколько из следующих: • <code>field</code> — текущий тип шаблона, например, <code>theming hook</code> (хук темизации); • <code>field-name-[имя_поля]</code> — имя текущего поля; например, если поле называется <code>field_description</code>, то получится значение <code>field-name-field-description</code>; • <code>field-type-[тип_поля]</code> — тип текущего поля; например, для имени поля <code>text</code> получится значение <code>field-type-text</code>; • <code>field-label-[позиция_метки]</code> — позиция текущей метки; например, при позиции <code>above</code> (вверху) получится значение <code>field-label-above</code>.
<code>\$element['#object']</code>	Сущность, к которой присоединено поле
<code>\$element['#view_mode']</code>	Режим просмотра сущности, к которой присоединено поле — <code>full</code> (полный) или <code>teaser</code> (тизер)
<code>\$element['#field_name']</code>	Имя поля
<code>\$element['#field_type']</code>	Тип поля
<code>\$element['#field_language']</code>	Язык поля
<code>\$element['#field_translatable']</code>	Является ли поле переводимым
<code>\$element['#label_display']</code>	Позиция вывода метки: <code>inline</code> (внутри строки), <code>above</code> (сверху) или <code>hidden</code> (скрытая)
<code>\$field_name_css</code>	Имя поля, совместимое с CSS
<code>\$field_type_css</code>	Тип поля, совместимый с CSS
<code>\$classes_array</code>	Массив значений атрибутов классов HTML; он же находится в виде строки в переменной <code>\$classes</code>

Файл `block.tpl.php`

Шаблон темизации уровня блоков `block.tpl.php` находится в каталоге `modules/block`. К этому времени вы уже должны уяснить типичную структуру файлов шаблонов.

```
<div id="<?php print $block_html_id; ?>"
  class="<?php print $classes; ?>"<?php print $attributes; ?>>

  <?php print render($title_prefix); ?>
<?php if ($block->subject): ?>
  <h2<?php print $title_attributes; ?>><?php print $block->subject ?></h2>
<?php endif;?>
  <?php print render($title_suffix); ?>
```

```

<div class="content"<?php print $content_attributes; ?>>
  <?php print $content ?>
</div>
</div>

```

Переменные, доступные по умолчанию в файле `block.tpl.php`, перечислены в табл. 9.7.

Таблица 9.7. Стандартные переменные, доступные в файле `block.tpl.php`

Переменная	Содержимое
<code>\$block->subject</code>	Заголовок блока
<code>\$content</code>	Содержимое блока
<code>\$block->module</code>	Модуль, сгенерировавший данный блок
<code>\$block->delta</code>	Идентификатор блока, уникальный в каждом модуле
<code>\$block->region</code>	Область, содержащая данный блок
<code>\$classes</code>	Строка классов, которые могут использоваться для контекстуальной стилизации с помощью CSS; ее можно изменять через переменную <code>\$classes_array</code> из функций препроцессора. Стандартные значения — одно или несколько из следующих: <code>block</code> — текущий тип шаблона, например, <code>theming hook</code> (хук темизации); <code>block-[модуль]</code> — модуль, сгенерировавший блок; например, модуль пользователя отвечает за обработку стандартного блока навигации по пользователям — в этом случае классом будет <code>block-user</code>.
<code>\$title_prefix</code> (массив)	Массив с дополнительными выходными данными модулей, которые нужно вывести перед дескриптором главного заголовка в шаблоне
<code>\$title_suffix</code> (массив)	Массив с дополнительными выходными данными модулей, которые нужно вывести сразу после дескриптора главного заголовка в шаблоне
<code>\$classes_array</code> (массив)	Массив значений атрибутов классов HTML; он же находится в виде строки в переменной <code>\$classes</code>
<code>\$block_zebra</code>	Возвращает либо <code>even</code> (четное), либо <code>odd</code> (нечетное) и используется для чересполосного вывода в списке тизеров
<code>\$block_id</code>	Зависит от области блока
<code>\$id</code>	То же, что и <code>\$block_id</code> , но не зависит от области блока
<code>\$is_front</code>	Флаг, равный <code>TRUE</code> , если данная страница является домашней страницей сайта
<code>\$logged_in</code>	Флаг, равный <code>TRUE</code> , если текущий пользователь является вошедшим зарегистрированным пользователем
<code>\$is_admin</code>	Флаг, равный <code>TRUE</code> , если текущий пользователь является администратором
<code>\$block_html_id</code>	Допустимый и гарантированно уникальный HTML-идентификатор

Переопределение шаблонных файлов

Скорее всего, хотя бы иногда понадобится изменять способ вывода элементов на сайте с помощью `page.tpl.php`, `node.tpl.php` или какого-то другого шаблона. Сейчас мы вернемся к теме Grayscale, созданной в начале главы, и изменим файл `page.tpl.php`, чтобы выдавалось приветствие, если посетитель зашел на переднюю страницу сайта. Если посетитель находится на любой другой странице, сообщение отображаться не должно. Для начала скопируйте файл `page.tpl.php` из каталога `modules/system` в каталог `sites/all/themes/grayscale/templates`. В результате Drupal будет использовать именно эту версию шаблона `page.tpl.php`, а не версию из каталога `modules/system`.

Теперь немного изменим скопированный файл. С помощью переменной `$is_front`, доступной в шаблоне страницы, мы будем проверять, находится ли посетитель на передней странице, и в таких случаях выводить в верхней части страницы сообщение **Welcome to My Site** (Добро пожаловать на сайт). Откройте файл и найдите в нем такую строку:

```
<div id="content" class="column"><div class="section">
```

Сразу после нее вставьте следующий фрагмент кода, который проверяет с помощью переменной `$is_front`, находится ли посетитель на передней странице, и если да, то выводит приветствие:

```
<?php

if ($is_front): ?><div id="welcome_message">
  <?php print "Welcome to My Site!"; ?> </div>
<?php endif; ?>
```

Теперь сообщение **Welcome to My Site** будет выводиться непосредственно под вторичным меню. Ничего особенного, но данный пример демонстрирует концепции использования и настройки стандартного файла `page.tpl.php`.

На заметку! Если переделанный файл `page.tpl.php` никак не влияет на поведение сайта, зайдите на страницу `admin/config/development/performance` и очистите кэш сайта.

Собственные варианты `.tpl`-файлов можно создавать и для других страниц сайта. Например, можно скопировать файл `page.tpl.php` и создать файл `page--front.tpl.php`. Этот новый шаблон будет применяться только к передней странице сайта. То же самое можно сделать и с файлом `node.tpl.php`. Предположим, что вам понадобилось темизировать статьи не так, как другие типы нод, такие как базовая страница. Для этого можно скопировать файл `node.tpl.php` из каталога `modules/node` в каталог `theme` и переименовать этот файл в `node--article.tpl.php`. Этот новый временный файл переопределит стандартный файл `node.tpl.php` для всех нод, которые являются статьями. Более подробно об этом можно прочитать на сайте Drupal.org по адресу <http://drupal.org/node/925532>.

На заметку! Между `node` и `article` указаны два дефиса — это требование Drupal.

Другие шаблонные файлы

При просмотре содержимого каталогов модулей и тем в них можно заметить и другие шаблонные файлы. Например, модуль комментирования использует для отображения комментариев шаблон `comment.tpl.php`. Этот модуль создает ряд переменных

и предоставляет их файлу `comment.tpl.php`. Назначение файла `comment.tpl.php` в качестве шаблона комментариев осуществляется за счет передачи пары `'template' => 'comment'` в одном из элементов массива-параметра в вызове `hook_theme()` (см. приведенный ниже код). Расширение файла `.tpl.php` указывать не обязательно, т.к. Drupal предполагает его наличие. Остальные подробности создания переменных и предоставления их шаблону будут приведены чуть ниже.

```
/**
 * Реализация hook_theme().
 */
function comment_theme() {
  return array(
    'comment_block' => array(
      'variables' => array(),
    ),
    'comment_preview' => array(
      'variables' => array('comment' => NULL),
    ),
    'comment' => array(
      'template' => 'comment',
      'render element' => 'elements',
    ),
    'comment_post_forbidden' => array(
      'variables' => array('node' => NULL),
    ),
    'comment_wrapper' => array(
      'template' => 'comment-wrapper',
      'render element' => 'content',
    ),
  );
}
```

Знакомство с функцией `theme()`

Когда среде Drupal нужно сгенерировать некоторый HTML-вывод для элемента, допускающего темизацию (такого как нода, блок, навигационная цепочка, комментариев или подпись пользователя), она пытается найти функцию темы или шаблонный файл, который сгенерирует HTML-код для данного элемента. Почти все части Drupal поддерживают темизацию, т.е. можно переопределять действительный HTML-код, генерируемый для этого элемента. Вскоре мы рассмотрим несколько примеров.

Совет. Список элементов Drupal, допускающих темизацию, приведен по адресу <http://api.drupal.org/api/group/themeable/7>.

Принцип действия функции `theme()`

Ниже приведен высокоуровневый обзор действий, которые происходят при отображении простой страницы ноды, такой как `http://example.com/?q=node/3`.

1. Система меню Drupal получает запрос и передает управление модулю ноды.
2. После создания структуры данных для ноды выполняется вызов `theme('node', $variables)`. Он находит нужную функцию темы или шаблонный файл, опреде-

ляет переменные, которые могут быть использованы шаблоном, и применяет шаблон, после чего получает окончательный HTML для ноды. (При выводе нескольких нод, как в случае блога, этот процесс повторяется для каждой ноды.)

3. Структура HTML возвращается в виде переменной `$return` в файле `index.php` и снова передается функции `theme()` в вызове `theme('page', $return)`.
4. Перед обработкой шаблона страницы Drupal предварительно определяет, какие области доступны и какие блоки следует выводить в каждой области. Каждый блок превращается в HTML-разметку с помощью вызова `theme('blocks', $region)`, который определяет переменные и применяет шаблон блока. Наверняка вы уже уловили общую схему обработки.
5. И, наконец, Drupal определяет переменные для шаблона страницы и применяет этот шаблон страницы.

Из этого списка должно быть понятно, что функция `theme()` — весьма важный компонент Drupal. Она отвечает за выполнение функций предварительной обработки (в них устанавливаются переменные, которые будут использоваться в шаблонах), за передачу вызова темы нужной функции и за поиск необходимого шаблонного файла. В результате получается HTML-код. Ниже мы подробно рассмотрим работу этой функции, а пока достаточно знать, что когда требуется преобразовать ноду в HTML-разметку, вызывается `theme('node', $variables = array())`. В зависимости от активной темы HTML-разметка генерируется функцией `theme_node()` или шаблонным файлом по имени `node.tpl.php`.

Этот процесс может быть переопределен на многих уровнях. Например, темы могут переопределить действие встроенных функций тем, поэтому при вызове `theme('node', $variables = array())` вместо `theme_node()` может отработать функция `grayscale_node()`. Для шаблонных файлов существуют соглашения по именованию, которые мы рассмотрим чуть ниже, и шаблонный файл `node--story.tpl.php` будет применяться только к нодам типа `Story`.

Переопределение элементов, допускающих темизацию

Как было показано выше, элементы, допускающие темизацию, распознаются по именам их функций, которые начинаются с префикса `theme_`, или по присутствию шаблонного файла (список всех стандартных элементов, допускающих темизацию, можно посмотреть по адресу <http://api.drupal.org/api/group/themeable/7>). Это соглашение по именованию позволяет создать в Drupal механизм переопределения для всех функций, допускающих темизацию. Дизайнеры могут указать на необходимость выполнения Drupal альтернативной функции, которая имеет преимущество перед функциями тем, предоставляемыми разработчиками модулей, или стандартными шаблонными файлами. Для примера рассмотрим работу этого процесса при создании навигационной цепочки сайта.

Откройте файл `includes/theme.inc` и просмотрите находящиеся в нем функции. Многие из них начинаются с префикса `theme_`, который означает возможность их переопределения. В частности, рассмотрим функцию `theme_breadcrumb()`:

```
/**
 * Возвращает HTML-разметку для навигационной цепочки.
 *
 * @param $variables
 *   Ассоциативный массив, содержащий:
 *   - breadcrumb — массив, содержащий навигационные ссылки.
 */
```

```
function theme_breadcrumb($variables) {
  $breadcrumb = $variables['breadcrumb'];
  if (!empty($breadcrumb)) {
    // Предоставление контекста для навигационных ссылок пользователям,
    // читающим с экрана. Заголовок можно сделать видимым
    // с помощью .element-invisible.
    $output = '<h2 class="element-invisible">' . t('You are here') . '</h2>';
    $output .= '<div class="breadcrumb">' . implode(' » ', $breadcrumb) . '</div>';
    return $output;
  }
}
```

Эта функция управляет генерацией HTML-разметки для навигационной цепочки в Drupal. В данном случае она добавляет разделитель — двойную стрелку вправо — между всеми элементами цепочки. Предположим, что нам понадобилось заменить дескриптор `div` на `span`, а вместо двойных стрелок использовать звездочки. Как это сделать? Один из способов — изменить эту функцию в файле `theme.inc` и сохранить ее. (Только не *поступайте* так на самом деле!) Однако существуют более удачные решения.

Видели ли вы, как эти функции тем вызываются в рамках ядра? Функция `theme_breadcrumb()` никогда не вызывается напрямую, а упаковывается во вспомогательную функцию `theme()`. Можно ожидать, что функция вызывается следующим образом:

```
theme_breadcrumb($variables)
```

Однако это не так. Разработчики используют такой вызов:

```
theme('breadcrumb', $variables);
```

Эта обобщенная функция `theme()` отвечает за инициализацию уровня тем и диспетчеризацию вызовов функций в нужные места, что приводит к более элегантному решению задачи. Вызов `theme()` инструктирует Drupal искать функции цепочки в описанном ниже порядке.

Если применяется тема *Grayscale*, основанная на *PHPTemplate*, Drupal ищет такую последовательность (пока что проигнорируем `breadcrumb.tpl.php`):

```
grayscale_breadcrumb()
sites/all/themes/grayscale/breadcrumb.tpl.php
theme_breadcrumb()
```

То есть файл темы `template.php` — место для переопределения стандартных функций тем Drupal, а также для перехвата и создания собственных переменных для их передачи в шаблонные файлы.

На заметку! При выполнении этих упражнений не делайте тему *Bartik* активной темой, т.к. у нее уже есть файл `template.php`. Используйте вместо нее тему *Grayscale* или *Stark*.

Для настройки навигационных цепочек Drupal создайте (либо измените уже созданный) файл `sites/all/themes/grayscale/template.php` и скопируйте в него функцию `theme_breadcrumb()` из файла `theme.inc`. Не забудьте вставить начальный дескриптор `<?php`. Кроме того, измените имя функции `theme_breadcrumb` на `grayscale_breadcrumb`. Затем щелкните на ссылке **Modules (Модули)** в меню в верхней части страницы для пересоздания реестра тем, чтобы среда Drupal обнаружила новую функцию.

```
<?php
/**
 * Возвращает HTML для навигационной цепочки.
 *
 * @param $variables
```

```

*   Ассоциативный массив, содержащий:
*   - breadcrumb — массив, содержащий навигационные ссылки.
*/
function grayscale_breadcrumb($variables) {
    $breadcrumb = $variables['breadcrumb'];
    if (!empty($breadcrumb)) {
        // Предоставление контекста для навигационных ссылок пользователям,
        // читающим с экрана. Заголовок можно сделать видимым
        // с помощью .element-invisible.
        $output = '<h2 class="element-invisible">' . t('You are here') . '</h2>';
        $output .= '<div class="breadcrumb">' . implode(' * ', $breadcrumb) . '</div>';
        return $output;
    }
}

```

После этого, если вы используете тему Grayscale, добавьте в файл `css/style.css` следующий фрагмент:

```

.breadcrumb {
    margin-top: 10px;
    clear: both;
    height: 15px;
    background-color: #fff;
    width: 960px;
    margin-right: auto;
    margin-left: auto;
}

```

Когда в следующий раз среде Drupal понадобится сформировать навигационную цепочку, она найдет сначала нашу функцию и применит ее, а не стандартную функцию `theme_breadcrumb()`, и цепочки будут содержать не двойные стрелки, а звездочки. Ловко, да? Пропуская все вызовы функций тем через функцию `theme()`, Drupal всегда проверяет, переопределяет ли текущая тема какую-то из функций с префиксом `theme_`, и если да, то вызывает ее.

На заметку! Все части ваших модулей, которые выводят HTML- или XML-код, должны выполняться только внутри функций тем, чтобы их вызовы можно было переопределять.

Переопределение с помощью шаблонных файлов

При совместной работе с дизайнером советы наподобие “найди в коде, где находятся функции, допускающие темизацию, и переопредели их” исключены. К счастью, имеется другой способ, более доступный именно дизайнерам. Элементы, допускающие темизацию, можно отобразить на собственные шаблонные файлы. Рассмотрим этот прием на примере с навигационной цепочкой.

Вначале проверьте, что ни одна из функций тем не переопределяет `theme_breadcrumb()`. Если вы при чтении предыдущего раздела создали в файле `template.php` своей темы функцию `grayscale_breadcrumb()`, закомментируйте ее. Затем создайте файл `sites/all/themes/grayscale/breadcrumb.tpl.php`. Это будет специальный шаблонный файл для навигационных цепочек. Поскольку нам нужно заменить дескриптор `<div>` на `<span>`, поместите в данный файл следующий код:

```

<?php if (!empty($breadcrumb)): ?>
    <span class="breadcrumb"><?php print implode(' ! ', $breadcrumb) ?></span>
<?php endif; ?>

```

Такой текст дизайнер может изменить без особого труда. Теперь нужно указать, чтобы среда Drupal при отображении цепочки вызывала этот шаблонный файл. Для этого очистите файлы кэша сайта, чтобы заново создать реестр тем: зайдите на страницу `admin/config/development/performance` и щелкните на ссылке `Clear all caches` (Очистить все кэши). При перестроении реестра темы Drupal обнаружит файл `breadcrumb.tpl.php` и отобразит элемента цепочки, допускающий темизацию, на этот шаблонный файл.

Теперь вы знаете, как переопределять любой элемент, допускающий темизацию, в Drupal так, чтобы упростить жизнь дизайнерам.

Добавление и манипулирование шаблонными переменными

Сейчас мы рассмотрим манипулирование или добавление переменных, которые передаются в шаблоны страницы и ноды. Вернемся к нашему примеру с навигационной цепочкой. Вначале давайте модифицируем файл `sites/all/themes/grayscale/breadcrumb.tpl.php`, чтобы использовать в качестве разделителя между элементами цепочки значение переменной `$breadcrumb_delimiter`:

```
<?php if (!empty($breadcrumb)): ?>
  <span class="breadcrumb">
    <?php print implode(' '. $breadcrumb_delimiter . ' ', $breadcrumb) ?>
  </span>
<?php endif; ?>
```

Установить переменную `$breadcrumb_delimiter` можно в модуле. Создадим для этого файл `sites/all/modules/crumbpicker.info`:

```
name = Breadcrumb Picker
description = Provide a character for the breadcrumb trail delimiter.
package = Pro Drupal Development
core = 7.x
```

Модуль `sites/all/modules/crumbpicker.module` будет небольшим:

```
<?php
/**
 * @file
 * Предоставление символа для разделителя элементов навигационной цепочки.
 */
/**
 * Реализация $module_name_preprocess_hook().
 */
function crumbpicker_preprocess_breadcrumb(&$variables) {
  $variables['breadcrumb_delimiter'] = '/';
}
```

После активации модуля навигационная цепочка будет иметь вид `Home / Administer / Site building`.

В приведенном выше примере демонстрируется установка в модуле переменной, которая используется в шаблонном файле. Однако создавать модуль каждый раз, когда переменная должна быть установлена, слишком сложно. Конечно же, нас выручит `template.php`. Напишем функцию для задания разделителя элементов цепочки.

Добавьте в файл `template.php` следующий код:

```
/**
 * Реализация $themeengine_name_preprocess_hook().
 * Устанавливаемые здесь переменные будут доступны в шаблонном файле
 * навигационной цепочки.
```

```
*/
function grayscale_preprocess_breadcrumb(&$variables) {
  $variables['breadcrumb_delimiter'] = '#';
}
```

Разумеется, это проще создания модуля, и такой способ обычно лучше всего подходит для передачи значений переменных из модулей в шаблоны. Все-таки модули обычно не пишутся только для этой цели. Итак, теперь у нас имеется модуль, предоставляющий переменную, и функция в файле `template.php`, которая тоже предоставляет такую переменную. Какая из переменных будет использована?

В действительности вся иерархия препроцессорных функций выполняется в определенном порядке, и каждая может переопределить значения переменных, определенных предыдущими препроцессорными функциями. В последнем примере разделителем элементов цепочки будет #, т.к. функция `phptemplate_preprocess_breadcrumb()` выполняется после `crumbpicker_preprocess_breadcrumb()`, и ее присваивания переменных переопределяют все присваивания для `$breadcrumb_delimiter`.

В случае темизации навигационной цепочки с использованием Grayscale порядок выполнения (от первого до последнего вызова) выглядит следующим образом:

```
template_preprocess()
template_preprocess_breadcrumb()
crumbpicker_preprocess()
crumbpicker_preprocess_breadcrumb()
phptemplate_preprocess()
phptemplate_preprocess_breadcrumb()
grayscale_preprocess()
grayscale_preprocess_breadcrumb()
template_process()
```

Значит, функция `grayscale_preprocess_breadcrumb()` может переопределить любую переменную, которая была ранее установлена, поскольку она вызывается последней перед передачей переменных шаблонному файлу. Такие вызовы могут показаться напрасной тратой времени, если реализована лишь небольшая часть этих функций. В общем-то, так оно и есть, и поэтому после построения реестра тем Drupal определяет, какие функции реализованы, и вызывает только их.

Использование модуля разработчика тем

Бесценным ресурсом при работе с темами Drupal является модуль разработчика тем. Он является частью модуля `devel.module`, и его можно загрузить по адресу http://drupal.org/project/devel_themer. Модуль разработчика тем позволяет указать на элемент на странице и узнать, какие шаблоны или функции тем задействованы при создании этого элемента, а также какие переменные (и их значения) доступны для него.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Создавать шаблонные файлы.
- Переопределять функции тем.
- Манипулировать переменными шаблонов.
- Создавать новые страничные области для блоков.

Дополнительную информацию о темах в Drupal 7 можно почерпнуть на странице справочника по темам по адресу <http://drupal.org/node/925532>.

ГЛАВА 10

Работа с блоками

Блоки — это фрагменты текста или функциональности, которые можно разместить в областях, определенных в шаблоне. Блоком может быть что угодно: одиночная нода, список нод, календарь, видеоклип, форма, онлайн-опрос, окно обмена сообщениями, поток изменений статусов в Facebook — в общем, практически все, что только можно себе представить. Когда речь идет о блоках, клиенты часто переспрашивают: “Значит, блок — это просто графический элемент?” Действительно, этот термин часто употребляется за пределами сферы Drupal для элементов, которые представляются в Drupal в виде блоков. В настоящей главе вы узнаете, как создавать собственные блоки и использовать их в своем сайте Drupal.

Что такое блок?

Как уже было сказано, блок — это обособленный контейнер, который может содержать практически все, что угодно. Проще всего понять, что такое блок, рассмотрев несколько примеров. В списке, приведенном в табл. 10.1, перечислены некоторые стандартные блоки, поставляемые вместе с Drupal.

Таблица 10.1. Стандартные блоки

Блок	Назначение
Login form (Входная форма)	Форма, отображаемая на сайте, которая позволяет пользователю выполнить вход, зарегистрировать новую учетную запись или изменить свой пароль
Who's online (Сейчас на сайте)	Блок со списком всех пользователей, которые в данный момент вошли на сайт
Who's new (Новые пользователи)	Блок со списком самых новых пользователей веб-сайта
Search form (Форма поиска)	Форма поиска, содержащаяся в блоке
Recent comments (Последние комментарии)	Список самых последних комментариев, отправленных на сайт
Main menu (Главное меню) и Secondary menu (Вторичное меню)	Оба меню доступны в виде блоков
Recent content (Последнее содержимое)	Список самых последних нод, отправленных на сайт
Most recent poll (Последний опрос)	Самый последний опрос на сайте отображается как блок (требуется активный модуль Poll)
Active forum topics (Активные темы форума)	Список тем форума, активных в последнее время (требуется активный модуль Forum)

Многие специальные модули включают блоки в качестве компонента обеспечиваемого ими функционального решения. Например, модуль Ubercart предоставляет несколько блоков для отображения такой информации, как состояние корзины посетителя.

Block API и интерфейс администрирования блоков позволяют создавать собственные блоки практически для любых целей. В табл. 10.2 приведены примеры специальных блоков, созданные автором за последнее время.

Таблица 10.2. Примеры специальных блоков

Блок	Назначение
Recent Bloggers (Последние блогеры)	Блок с галереей аватаров блогеров, которые разместили на сайте последние сообщения
Slideshow of upcoming events (Слайд-шоу из предстоящих событий)	Блок, отображающий тизеры нод для предстоящих событий в виде слайд-шоу
A chat form (Форма для чата)	Блок, который содержит графический элемент для чата Meebo и выводит этот элемент в левой или правой врезке
A donate now feature (Функция добровольного взноса)	Блок с кнопкой, которая позволяет пользователю внести взнос с помощью платежной системы PayPal
A list of new books added to a library's collection (Список новых книг, добавленных в библиотеку)	Блок с мини-изображением обложки книги, названием и автором книги и кнопкой "Зарезервировать"
A contact us form (Форма "Контакты")	Блок с простой формой запроса контактных данных
A list of postings on multiple social networking sites (Список сообщений на нескольких сайтах социальной сети)	Блок с лентами из нескольких сайтов социальной сети в виде общего смешанного списка
A Google map showing recent postings (Карта Google, показывающая последние сообщения)	Блок, выводящий маркеры на карте Google для нод, которые содержат географические координаты

Блоки определяются либо с помощью веб-интерфейса Drupal (рис. 10.1), либо программно через Block API (блоки, предоставляемые модулями). А как узнать, какой метод использовать для создания блока? Одноразовый блок вроде фрагмента статического HTML, относящегося к сайту — хороший кандидат на роль специального блока. Динамические по своей природе блоки, которые относятся к написанному вами модулю или состоят в основном из PHP-кода — хорошие кандидаты на применение Block API с реализацией внутри модуля. Старайтесь не держать PHP-код в собственных блоках, т.к. сопровождать код, хранящийся в базе данных, сложнее, чем код, размещенный в модуле. К тому же редактор сайта может нечаянно удалить результаты вашей напряженной работы. Если нет смысла создавать блок на уровне модуля, просто вставьте в блок вызов собственной функции и разместите весь PHP-код где-то в другом месте.

Совет. При необходимости создать специальные блоки или другие компоненты для конкретного сайта обычно лучше написать специальный модуль и поместить в него всю необходимую функциональность. Например, разработчик веб-сайта для компании Jones Pies and Soda может создать модуль `jonespiesandsoda`.

Интерфейс Block API использовать легко, однако это не значит, что он не позволяет создавать сложные вещи. Блоки могут отображать практически все (поскольку пишутся на PHP и поэтому не ограничены в своих возможностях), но они обычно играют вспомо-

гательную роль по отношению к основному контенту сайта. Например, можно создать специальные навигационные блоки для каждой пользовательской роли или блок для вывода списка комментариев, ожидающих утверждения.

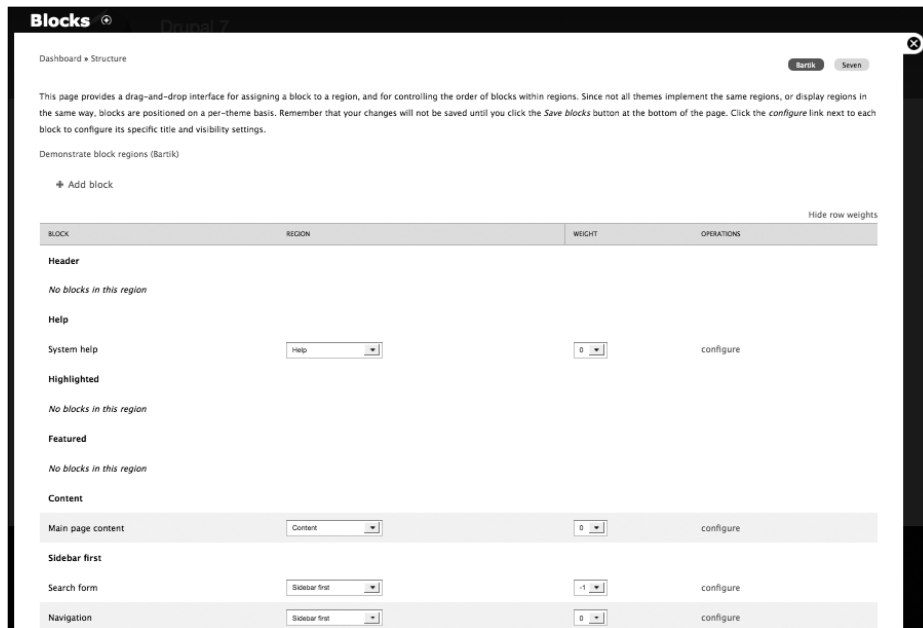


Рис. 10.1. Страница со списком блоков

Параметры настройки блоков

Часто применяемая настройка, с которой стоит внимательно ознакомиться — это параметры видимости блока на странице настройки этого блока. Видимость блока определяет, когда блок должен быть видимым на странице, а когда нет, в зависимости от критериев, заданных в интерфейсе, который показан на рис. 10.2. Рассмотрим в качестве примера блок User Login (Вход пользователя). Его отображением можно управлять с помощью перечисленных ниже параметров.

- *Параметры видимости на отдельных страницах.* Администраторы могут указать, что блок должен быть видимым или скрытым на конкретной странице или диапазоне страниц, или когда специальный фрагмент PHP-кода определит истинность каких-то условий.
- *Параметры видимости типов контента.* Администраторы могут указать, что блок должен быть видимым только на страницах, которые выводят конкретный тип контента — например, блок видим, только если страница содержит тему обсуждения.
- *Параметры видимости, зависящие от ролей.* Администраторы могут указать, что блок должен быть видимым только для пользователей с определенными ролями.
- *Параметры видимости, зависящие от пользователей.* Администраторы могут разрешить отдельным пользователям настраивать видимость некоторых блоков в параметрах учетных записей этих пользователей. Пользователи щелкают на ссылке My account (Моя учетная запись) и настраивают видимость блока.

Рис. 10.2. Экран настройки блока в интерфейсе администрирования

Размещение блока

Выше уже было сказано, что страница администрирования блоков позволяет администратору сайта указать области, где могут появляться определенные блоки. На той же странице можно задать и порядок вывода блоков в области (см. рис. 10.1). Области определяются уровнем тем в `.info`-файле темы, а не с помощью Block API, и разные темы могут отображать различные области. Создание областей описано в главе 8.

Определение блока

Блоки определяются в модулях с помощью вызова `hook_block()`, и внутри одного этого хука модуль может реализовать несколько блоков. После определения блока он будет отображаться на странице администрирования блоков. Кроме того, администратор сайта может вручную создать специальные блоки, используя веб-интерфейс. В данном разделе основное внимание будет уделено программному созданию блоков. Посмотрим на схему базы данных для блоков, приведенную на рис. 10.3.

Свойства каждого блока хранятся в таблице `block`. Дополнительные данные для блоков, которые созданы с помощью интерфейса настройки блока, хранятся в других сопровождающих таблицах, также показанных на рис. 10.3.

block	block_custom
bid module delta theme status weight region custom visibility pages title cache	bid body info format
	block node type
	module delta type
	block role
	module delta rid

Рис. 10.3. Схема базы данных для блоков

Ниже перечислены свойства, определенные в столбцах таблицы `block`.

- `bid`. Уникальный идентификатор каждого блока.
- `module`. Содержит имя модуля, в котором определен данный блок. Блок входа пользователя создан в модуле `user` и т.д. Специализированные блоки, созданные администратором на странице `Structure⇒Blocks⇒Add Blocks` (Структура⇒Блоки⇒Добавить блоки), считаются созданными модулем `block`.
- `delta`. В одном вызове `hook_block()` модуль может определить несколько блоков, и поэтому столбец `delta` хранит ключ для каждого блока, уникальный лишь для каждой реализации `hook_block()`, но не для всех блоков вообще. Значение `delta` может быть целым числом или строкой.
- `theme`. Блоки можно определить для нескольких тем. Поэтому необходимо хранить имя темы, для которой активен блок. Для каждой такой темы в базе данных предусмотрена отдельная строка. Параметры конфигурации не распространяются на все темы.
- `status`. Содержит информацию о том, активирован ли блок. Значение 1 означает, что активирован, а значение 0 — отключен. Если с блоком не связана никакая область, Drupal заносит в это поле значение 0.
- `weight`. Вес блока определяет его позицию относительно других блоков в области.
- `region`. Имя области, в которой должен появиться блок — например, `footer`.
- `custom`. Возможность настройки видимости данного блока пользователем (см. рис. 10.3). Значение 0 означает, что пользователь не может управлять видимостью данного блока; значение 1 — что блок по умолчанию видим, но пользователь может скрыть его; 2 — что блок по умолчанию скрыт, но пользователь может сделать его видимым.
- `visibility`. Определяет видимость блока. Значение 1 означает, что блок видим на всех страницах, кроме перечисленных; 1 — что блок видим только на перечисленных страницах; 2 — что для определения видимости Drupal должен выполнить специальный PHP-код, заданный администратором.

- `pages`. Содержимое этого поля зависит от значения поля `visibility`. Если это значение равно 0 или 1, то это поле содержит список путей Drupal. Если значение `visibility` равно 2, то поле `pages` содержит специальный PHP-код, который необходимо выполнить, чтобы определить, нужно ли отображать блок.
- `title`. Пользовательский заголовок для блока. Если данное поле пусто, используется стандартный заголовок блока (из модуля, предоставляющего этот блок). Если оно содержит `<none>`, то заголовок для блока выводиться не будет. Иначе текст из данного поля будет использоваться в качестве заголовка блока.
- `cache`. Определяет способ кэширования данного блока. Значение -1 означает, что блок не должен кэшироваться; значение 1 — кэширование блока для каждой роли (значение по умолчанию в Drupal для блоков, в которых не указан параметр кэширования); значение 2 — кэширование блока для каждого пользователя; значение 4 — кэширование для каждой страницы. Значение 8 означает, что блок будет кэшироваться одинаково для всех ролей, пользователей и страниц.

Использование блочных хуков

Блочные хуки — `hook_block_info()`, `hook_block_configure()`, `hook_block_save()` и `hook_block_view()` — управляют всей логикой программного создания блоков. Использование этих хуков позволяет объявить один блок или целый набор блоков. Функцию `hook_block()` для создания блоков можно реализовать в любом модуле. Рассмотрим все эти хуки.

- `hook_block_info()` — определяет все блоки, предоставляемые модулем.
- `hook_block_configure($delta = '')` — форма настройки для блока. Параметр `$delta` представляет собой идентификатор возвращаемого блока и может быть целочисленным или строковым. Этот же параметр используется в хуках `hook_block_save` и `hook_block_view`.
- `hook_block_save($delta = '', $edit = array())` — сохраняет параметры конфигурации для блока. Параметр `$edit` содержит данные, отправленные из формы настройки для блока.
- `hook_block_view($delta = '')` — обрабатывает блок при активации в области, чтобы вывести его содержимое.

Создание блоков

В приведенном ниже примере мы создадим два блока, предназначенные для облегчения модерирования контента. Вначале нужно создать блок для списка комментариев, которые ожидают утверждения, а затем создать блок для списка неопубликованных нод. Оба блока будут содержать ссылки на форму редактирования для каждого фрагмента модерлируемого контента.

Для размещения кода блоков создадим новый модуль с именем `approval.module`. Создайте в каталоге `sites/all/modules/custom` новую папку с именем `approval` (если папки `modules` и `custom` не существуют, понадобится создать и их).

Теперь добавим в эту папку файл `approval.info`:

```
name = Approval
description = Blocks for facilitating pending content workflow.
package = Pro Drupal Development
core = 7.x
version = VERSION
files[] = approval.module
```

Затем добавим файл `approval.module`:

```
<?php

/**
 * @file
 * Реализация различных блоков для рабочего потока обработки ожидающего контента.
 */
```

После создания этих файлов активируйте модуль со страницы Modules (Модули). Нам еще предстоит поработать с файлом `approval.module`, поэтому не закрывайте текстовый редактор.

Добавим текст хука `block_hook_info`, чтобы блок появился в списке блоков на странице администрирования блоков (рис. 10.4). Определим заголовок блока с помощью атрибута `info`, состояние — `True` (автоматически включен), область — `sidebar_first`, вес равен 0, а видимость — 1 (видим):

```
/**
 * Реализация hook_block_info().
 */
function approval_block_info() {

  $blocks['pending_comments'] = array(
    'info'           => t('Pending Comments'), // Ожидающие комментарии
    'status'         => TRUE,
    'region'         => 'sidebar_first',
    'weight'         => 0,
    'visibility'     => 1,
  );
  return $blocks;
}
```

Обратите внимание, что значение `info` — это не заголовок блока, который отображается пользователям, если блок активен; это описание, выводимое только в списке блоков, доступных администратору для настройки. А сам заголовок блока будет реализован позже. Но сначала нужно задать дополнительные параметры настройки. Для этого понадобится реализовать функцию `hook_block_configure`, приведенную в следующем кодовом фрагменте. Мы создадим новое поле формы, видимой после щелчка на ссылке `configure` (настройка) рядом с блоком на странице администрирования блоков (рис. 10.5).

Sidebar first			
Search form	Sidebar first	-1	configure
Navigation	Sidebar first	0	configure
Pending Comments	Sidebar first	0	configure
User login	Sidebar first	0	configure
Management	Sidebar first	1	configure

Рис. 10.4. Теперь блок Pending comments (Ожидающие комментарии) находится на странице обзора блоков под заголовком области Sidebar First (Первая врезка)

'Pending Comments' block

Block title

Override the default title for the block. Use *<none>* to display no title, or leave blank to use the default block title.

Configure Number of Comments

5

Enter the number of pending comments that will appear in the block.

Рис. 10.5. Форма настройки блока со специальными полями

```

/**
 * Реализация hook_block_configure().
 */
function approval_block_configure($delta) {
    $form = array();
    switch($delta) {
        case 'pending_comments':
            $form['pending_comment_count'] = array(
                '#type' => 'textfield',
                '#title' => t('Configure Number of Comments to Display'),
                // Укажите количество выводимых комментариев
                '#size' => 6,
                '#description' => t('Enter the number of pending comments that will
appear in the block.'),
                // Введите количество ожидающих комментариев,
                // которые будут отображаться в блоке.
                '#default_value' => variable_get('pending_comment_count', 5),
            );
            break;
    }

    return $form;
}

```

После отправки формы настройки блока, показанной на рис. 10.5, должен сработать хук `hook_block_save()`. Его можно использовать для сохранения значения поля формы:

```

/**
 * Реализация hook_block_save().
 */
function approval_block_save($delta = '', $edit = array()) {
    switch($delta) {
        case 'pending_comments':
            variable_set('pending_comment_count', (int)$edit['pending_comment_count']);
            break;
    }
    return;
}

```

Количество ожидающих комментариев, которое следует выводить, сохраняется с помощью встроенной системы переменных Drupal, а именно — с помощью функции `variable_set()`. Обратите внимание на приведение значения к целому виду в качестве проверки чистоты данных.

И, наконец, добавим операцию просмотра с помощью хука `hook_block_view` и пользовательской функции, которая возвращает список ожидающих комментариев при отображении блока:

```
/**
 * Реализация hook_block_view().
 */
function approval_block_view($delta = '') {

  switch ($delta) {
    case 'pending_comments':
      $block['subject'] = t('Pending Comments'); // Ожидающие комментарии
      $block['content'] = approval_block_contents($delta);
      return $block;
      break;
  }
}

/**
 * Функция отображения контента блока, определенная в модуле.
 */
function approval_block_contents($delta) {
  switch ($delta) {
    case 'pending_comments':
      if (user_access('administer comments')) {
        $nbr_comments = variable_get('pending_comment_count');
        $result = db_query("SELECT cid, subject FROM {comment} WHERE status = 0
          limit $nbr_comments");
        $items = array();
        foreach ($result as $row) {
          $items[] = l($row->subject, 'comment/' . $row->cid . '/edit');
        }
        return array('#markup' => theme('item_list', array('items' => $items)));
      }
      break;
  }
}
```

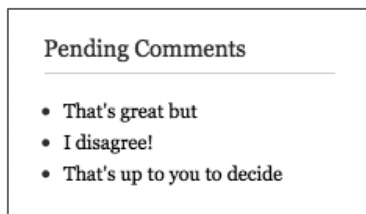


Рис. 10.6. Блок со списком ожидающих комментариев после его активации; содержит три ожидающих комментария

Здесь выполняется запрос из базы данных комментариев, которые ожидают утверждения, и вывод заголовков этих комментариев в качестве ссылок на сами комментарии, как показано на рис. 10.6.

Заголовок блока задается следующей строкой:

```
$block['subject'] = t('Pending comments');
// Ожидающие комментарии
```

Теперь блок вывода ожидающих комментариев завершен, и в этой функции `approval_block()` можно определить другой блок — для вывода всех неопубликованных нод со ссылками на страницу их редактирования:


```

/**
 * Реализация hook_block_info().
 */
function approval_block_info() {
  $blocks['pending_comments'] = array(
    'info'      => t('Pending comments'), // Ожидающие комментарии
    'status'    => TRUE,
    'region'    => 'sidebar_first',
    'weight'    => 0,
    'visibility' => 1,
  );
  $blocks['unpublished_nodes'] = array(
    'info'      => t('Unpublished nodes'), // Неопубликованные ноды
    'status'    => TRUE,
    'region'    => 'sidebar_first',
    'weight'    => 0,
    'visibility' => 1,
  );
  return $blocks;
}

```

Обратите внимание на указание ключей для блоков: `$blocks['pending_comments']`, `$blocks['unpublished_nodes']`, ..., `$blocks['xxxxxxx']`. Модуль блоков будет использовать эти ключи по порядку в качестве параметра `$delta`.

Добавим к функциям `hook_block_configure` и `hook_block_save` форму для установки количества отображаемых нод и сохранения значения, введенного в форме администратором сайта:

```

/**
 * Реализация hook_block_configure().
 */
function approval_block_configure($delta) {
  $form = array();

  switch($delta) {

    case 'pending_comments':
      $form['pending_comment_count'] = array(
        '#type' => 'textfield',
        '#title' => t('Configure number of comments to display'),
        // Укажите количество выводимых комментариев
        '#size' => 6,
        '#description' => t('Enter the number of pending comments that will
appear in the block.'),
        // Введите количество ожидающих комментариев,
        // которые будут отображаться в блоке.
        '#default_value' => variable_get('pending_comment_count', 5),
      );
      break;

    case 'unpublished_nodes':
      $form['unpublished_node_count'] = array(
        '#type' => 'textfield',
        '#title' => t('Configure Number of Nodes to Display'),
        // Укажите количество выводимых нод
        '#size' => 6,

```

```

    '#description' => t('Enter the number of unpublished nodes that will
appear in the block.'),
        // Введите количество неопубликованных нод,
        // которые будут отображаться в блоке.
    '#default_value' => variable_get('unpublished_node_count', 5),
    );
    break;
}
return $form;
}
/**
 * Реализация hook_block_save().
 */
function approval_block_save($delta = '', $edit = array()) {
    switch($delta) {
        case 'pending_comments':
            variable_set('pending_comment_count', (int)$edit['pending_comment_count']);
            break;
        case 'unpublished_nodes':
            variable_set('unpublished_nodes_count', (int)$edit['unpublished_node_count']);
            break;
    }
    return;
}

```

После этого добавим в функции `hook_block_view` и `approval_block_content` возможность вывода неопубликованных нод:

```

/**
 * Реализация hook_block_view().
 */
function approval_block_view($delta = '') {
    switch ($delta) {
        case 'pending_comments':
            $block['subject'] = t('Pending Comments'); // Ожидающие комментарии
            $block['content'] = approval_block_contents($delta);
            return $block;
            break;
        case 'unpublished_nodes':
            $block['subject'] = t('Unpublished Nodes'); // Неопубликованные ноды
            $block['content'] = approval_block_contents($delta);
            return $block;
            break;
    }
}
/**
 * Функция отображения контента блока, определенная в модуле.
 */
function approval_block_contents($delta) {
    switch ($delta) {
        case 'pending_comments':
            if (user_access('administer comments')) {
                $nbr_comments = variable_get('pending_comment_count');
                $result = db_query("SELECT cid, subject FROM {comment} WHERE status = 0
                    limit $nbr_comments");
                $items = array();
            }
        }
    }
}

```

```

foreach ($result as $row) {
    $items[] = l($row->subject, 'comment/' . $row->cid . '/edit');
}
return array('#markup' => theme('item_list', array('items' => $items)));
}
break;

case 'unpublished_nodes':
    if (user_access('administer nodes')) {
        $nbr_nodes = variable_get('unpublished_node_count');
        $result = db_query("SELECT nid, title FROM {node} WHERE status = 0
            limit $nbr_nodes");
        $items = array();
        foreach ($result as $row) {
            $items[] = l($row->title, 'node/' . $row->nid . '/edit');
        }
        return array('#markup' => theme('item_list', array('items' => $items)));
    }
    break;
}
}
}

```

Результат работы только что созданного блока для отображения неопубликованных нод показан на рис. 10.7.

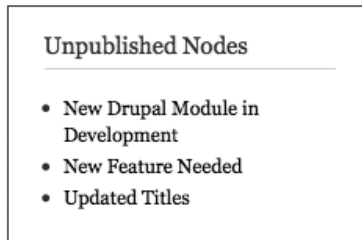


Рис. 10.7. Блок со списком неопубликованных нод

Активация блока при инсталляции модуля

В модуле `approval.module` мы автоматически активировали блоки и назначали им область темы. Например, при создании блока `Pending Comments` он был автоматически активирован (`status = TRUE`) и назначен области `sidebar_first`:

```

$blocks['pending_comments'] = array(
  'info'      => t('Pending Comments'), // Ожидающие комментарии
  'status'    => TRUE,
  'region'    => 'sidebar_first',
  'weight'    => 0,
);

```

В некоторых ситуациях требуется разрешить администратору сайта определять, активировать ли блок и какой области темы его назначить. В таких случаях понадобится установить атрибут `status` в `FALSE` и не назначать блоку никакой области. В следующем примере показано создание нового блока `Pending Users`, который не активируется автоматически, и которому не назначается область:

```
$blocks['pending_users'] = array(
  'info'      => t('Pending Users'), // Ожидающие пользователи
  'status'    => FALSE,
  'weight'    => 0,
);
```

Примеры видимости блоков

В интерфейсе администрирования блоков имеется раздел Page visibility settings (Параметры видимости страницы), где можно ввести фрагменты PHP-кода. При построении страницы Drupal выполняет такие фрагменты, чтобы определить, нужно ли выводить блок. Ниже приведены примеры наиболее ходовых фрагментов, каждый из которых возвращает значение TRUE или FALSE, означающее, что блок должен быть видимым или невидимым при данном запросе.

Отображение блока только для вошедших пользователей

Возвращает TRUE, только если значение `$user->uid` не равно 0:

```
<?php
global $user;
return (bool) $user->uid;
?>
```

Вывод блока только для анонимных пользователей

Возвращает TRUE, только если значение `$user->uid` равно 0:

```
<?php
global $user;
return !(bool) $user->uid;
?>
```

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Знать, что собой представляют блоки и чем они отличаются от нод.
- Знать, как работают параметры видимости и размещения блоков.
- Знать, как определить один или несколько блоков.
- Знать, как сделать блок активным по умолчанию.

ГЛАВА 11

Интерфейс Form API

Среда Drupal оснащена интерфейсом прикладного программирования (Application Programming Interface — API), который используется для генерации, проверки достоверности и обработки HTML-форм. Интерфейс Form API абстрагирует формы во вложенный массив свойств и значений. Затем этот массив генерируется как часть процесса построения разметки страницы, содержащей форму. Из такого подхода вытекают несколько следствий.

- Вместо того чтобы выводить HTML-код, мы создаем массив и позволяем механизму генерировать HTML-разметку.
- Поскольку мы имеем дело с представлением формы в виде структурированных данных, мы можем добавлять, удалять, переупорядочивать и изменять формы. Это особенно удобно, когда необходимо просто и ненавязчиво изменить форму, созданную другим модулем.
- Любой элемент формы можно отобразить на любую функцию темы.
- К любой форме могут быть добавлены дополнительные процедуры проверки достоверности или обработки формы.
- Операции с формами защищены против атак внедрением кода, когда пользователь изменяет информацию в полях формы и затем пытается отправить ее.

В этой главе будет представлен материал, касающийся начальной части стадии изучения. Вы узнаете, как работает механизм форм, как создавать формы, проверять их, обрабатывать и внедряться в процесс отправки формы. В этой главе описывается интерфейс Form API, реализованный в версии Drupal 7. Мы начнем с рассмотрения механизма обработки форм. Если вы только приступаете к работе с формами в Drupal и хотите начать с примера, возможно, стоит перейти сразу к разделу “Создание базовых форм”. Если вас интересуют детальные сведения об отдельных свойствах форм, вы найдете их в разделе “Свойства Form API”.

Обработка формы

На рис. 11.1 представлен обзор процессов создания, проверки достоверности и отправки формы. В последующих разделах этот рисунок будет использоваться в качестве руководства при описании того, что происходит во время упомянутых процессов.

Чтобы эффективно взаимодействовать с Form API, полезно знать механизм, обеспечивающий работу этого API-интерфейса. Модули описывают формы для Drupal, используя ассоциативные массивы. Механизм форм Drupal заботится о генерации HTML-кода для отображения форм, а также о безопасной обработке отправленных форм, используя три фазы: проверка достоверности, отправка и перенаправление. В последующих разделах рассказывается о том, что происходит, когда вызывается `drupal_get_form()`.

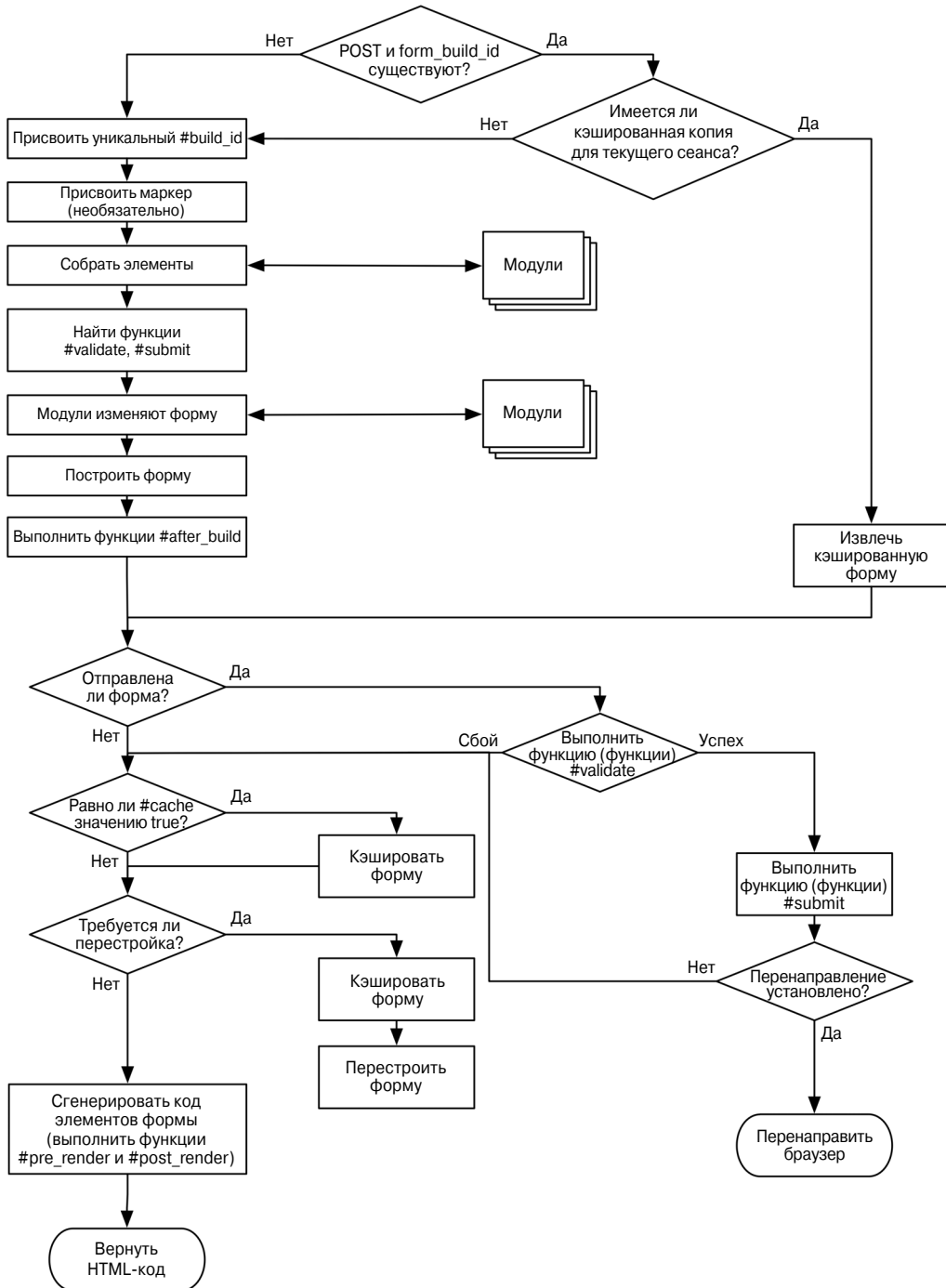


Рис. 11.1. Обработка форм в Drupal

Инициализация процесса

Существуют три переменных, которые очень важны при работе с формами. Первая из них — `$form_id` — содержит строку, идентифицирующую форму. Вторая — `$form` — это структурированный массив, описывающий форму. А третья — `$form_state` — содержит информацию о форме, такую как значения формы, и что должно произойти по завершении обработки формы. Работа функции `drupal_get_form()` начинается с инициализации `$form_state`.

Установка маркера

Одно из преимуществ системы форм состоит в том, что для безопасности и противодействия спамерам она старается гарантировать, что отправленная форма действительно является формой, созданной Drupal. Для этого Drupal устанавливает секретный ключ при каждой своей установке. Ключ генерируется случайным образом во время процесса установки и отличает эту конкретную установку Drupal от других ее установок. После генерации ключ хранится в таблице переменных как `drupal_private_key`. Псевдослучайный маркер, созданный на основе секретного ключа, передается в скрытом поле формы и проверяется при отправке формы. Основную информацию об этом можно найти по адресу <http://drupal.org/node/28420>. Маркеры используются только для зарегистрированных пользователей, в то время как страницы для анонимных пользователей обычно кэшируются, из-за чего получают неуникальные маркеры.

Установка идентификатора

Скрытое поле, содержащее идентификатор текущей формы, отправляется браузеру как часть формы. Этот идентификатор обычно соответствует функции, которая определяет форму, и передается в первом параметре `drupal_get_form()`. Например, функция `user_register()` определяет регистрационную форму пользователя и вызывается следующим образом:

```
$output = drupal_get_form('user_register');
```

Сбор всех возможных определений элемента формы

Затем вызывается функция `_element_info()`. Она запускает хук `hook_element_info()` на всех модулях, в которых он реализован. Внутри ядра Drupal стандартные элементы, такие как флажки и переключатели, определяются реализацией `module/system/system.module` функции `hook_element_info()`. Модули реализуют этот хук, если хотят определить собственные типы элемента. Вы могли бы реализовать `hook_element_info()` в своем модуле, чтобы наделить элемент формы специальными свойствами, такими как кнопка загрузки изображения, которая показывает миниатюру в течение предварительного просмотра ноды, или чтобы расширить существующий элемент формы, определив дополнительные свойства.

Например, сторонний модуль `fivestar` определяет собственный тип элемента:

```
/**
 * Реализация hook_elements().
 *
 * Определяет тип элемента формы 'fivestar'.
 */
function fivestar_element_info() {
  $type['fivestar'] = array(
```

```

    '#input'           => TRUE,
    '#stars'           => 5,
    '#widget'          => 'stars',
    '#allow_clear'     => FALSE,
    '#auto_submit'     => FALSE,
    '#auto_submit_path' => '',
    '#labels_enable'   => TRUE,
    '#process'         => array('fivestar_expand'),
  );
  return $type;
}

```

А модуль TinyMCE использует `hook_element_info()` для потенциальной модификации значений свойств по умолчанию существующего типа. TinyMCE добавляет свойство `#process` к типу элемента `textarea`, так что когда форма построена, будет вызвана функция `tinymce_process_textarea()`, которая может модифицировать этот элемент. Свойство `#process` — это массив имен функций для вызова.

```

/**
 * Реализация hook_elements().
 */
function tinymce_element_info() {
  $type = array();

  if (user_access('access tinymce')) {
    // Позволить TinyMCE потенциально обработать каждый элемент textarea.
    $type['textarea'] = array(
      '#process' => array('tinymce_process_textarea'),
    );
  }
  return $type;
}

```

Поиск функции проверки достоверности

Функция проверки достоверности для формы назначается установкой свойства `#validate` формы в массив с именем функции в качестве значения. Множество проверок достоверности можно определить следующим образом:

```

// Необходимо вызывать foo_validate() и bar_validate() во время проверки формы.
$form['#validate'][] = 'foo_validate';
$form['#validate'][] = 'bar_validate';

// Дополнительно поместить в форму значение, которое понадобится функции
// проверки, за счет создания уникального ключа в форме.
$form['#value_for_foo_validate'] = 'baz';

```

Если в форме нет свойства с именем `#validate`, следующий шаг состоит в поиске функции с именем, состоящим из идентификатора формы плюс `_validate`. Таким образом, если идентификатором формы является `user_register`, то свойство `#validate` будет установлено в `user_register_validate`.

Поиск функции отправки

Функция, которая обрабатывает отправку формы, назначается путем установки свойства `#submit` формы в массив с именем функции, которая будет отвечать за обработку отправки формы:


```
// Вызвать my_special_submit_function() для обработки отправки формы.
$form['#submit'][] = 'my_special_submit_function';

// Также вызвать my_second_submit_function().
$form['#submit'][] = 'my_second_submit_function';
```

Если нет свойства с именем `#submit`, Drupal пытается найти функцию с именем, образованным из идентификатора формы, дополненного `_submit`. Поэтому, если идентификатором формы является `user_register`, то среда Drupal устанавливает свойство `#submit` в функцию обработки формы, которую она находит, т.е. в `user_register_submit`.

Разрешение модулям изменять форму перед ее построением

Перед построением формы модули имеют два шанса изменить форму. Модули могут реализовать функцию, имя которой образуется из идентификатора формы с добавкой `_alter`, или просто реализовать `hook_form_alter()`. Любой модуль, в котором реализована та или другая функция, может изменять что-нибудь в форме. Это — первый способ модифицировать или переопределять формы, которые были созданы сторонними модулями.

Построение формы

Теперь форма передается в функцию `form_builder()`, которая рекурсивно обрабатывает ее через дерево формы и добавляет требуемые стандартные значения. Эта функция также проверяет ключ `#access` для каждого элемента и запрещает доступ к элементам формы и их дочерним элементам, если `#access` равно `FALSE` для этого элемента.

Разрешение функциям изменять форму после ее построения

Каждый раз, когда функция `form_builder()` обнаруживает новую ветвь дерева `$form` (например, ветвь с новым набором полей или новым элементом формы), она ищет свойство по имени `#after_build`. Это — необязательный массив функций, которые должны быть вызваны, как только будет построен текущий элемент формы. Когда построена вся форма, осуществляется финальный вызов необязательных функций, имена которых заданы в `$form['#after_build']`. Все функции `#after_build` получают значения `$form` и `$form_values` в виде параметров. Пример может служить отображение пути файловой системы в ядре на странице Configuration ⇒ File system (Конфигурация ⇒ Файловая система). Функция `#alter_build` (в данном случае — `system_check_directory()`) запускается для определения отсутствия каталога или отсутствия доступа к нему по записи и вывода признака ошибки напротив элемента формы — виновника проблемы.

Проверка факта отправки формы

Если вы внимательно посмотрите на рис. 11.1, то увидите, что мы достигли точки разветвления. Если форма отображается первый раз, Drupal создает HTML-код для формы. Если же форма отправляется, Drupal приступает к обработке данных, введенных пользователем на форме; мы возвратимся к этому моменту позже (раздел “Проверка достоверности формы” далее в главе). Пока что мы будем предполагать, что форма отображается впервые. Важно понимать, что вся работа, описанная ранее, касается и первого отображения формы, и отображения, когда форма отправляется.

Обнаружение функции темы для формы

Если свойство `$form['#theme']` установлено в существующую функцию, Drupal просто использует ее для темизации формы. В противном случае проверяется реестр тем на предмет наличия записи, соответствующей идентификатору этой формы. Если такая запись найдена, идентификатор формы присваивается `$form['#theme']`, чтобы позднее, при визуализации формы, была найдена функция темы на основе идентификатора формы. Например, если идентификатором формы является `taxonomy_overview_terms`, то Drupal вызовет соответствующую функцию темы — `theme_taxonomy_overview_terms()`. Разумеется, эта функция темы может быть переопределена функцией темы или файлом шаблона в специальной пользовательской теме; в главе 9 применение тем рассматривается более подробно.

Разрешение модулям изменять форму перед ее визуализацией

Единственное, что мы еще не рассмотрели — это преобразование формы из структуры данных в HTML-код. Но до того как это произойдет, модули имеют последний шанс провести дополнительную настройку. Это полезно для мастера многостраничных форм или в других случаях, когда нужно изменить форму в последнюю минуту. Любая функция, определенная в свойстве `form['#pre_render']`, вызывается и получает форму, подлежащую визуализации.

Визуализация формы

Чтобы преобразовывать дерево формы из вложенного массива в HTML-код, построитель формы вызывает `drupal_render()`. Эта рекурсивная функция проходит через все уровни дерева формы и на каждом из них выполняет перечисленные ниже действия.

1. Выясняет, был ли определен элемент `#children` (ассоциированный с контентом, сгенерированным для этого элемента); в противном случае отображает дочерний элемент этой ноды дерева следующим образом.
 - Выясняет, определена ли функция `#theme` для этого элемента.
 - Если функция `#theme` определена, временно устанавливает `#type` этого элемента в состояние `markup` (разметка). Затем передает этот элемент функции темы и возвращает элемент в состояние, которое было раньше.
 - Если никакого контента не генерировалось (либо из-за того, что не было функций `#theme`, определенных для этого элемента, либо потому, что в результате вызова функции `#theme` ничего не возвращалось), каждый из дочерних элементов этого элемента отображается поочередно (т.е. передачей дочернего элемента функции `drupal_render()`).
 - С другой стороны, если контент был сгенерирован функцией `#theme`, он запоминается в свойстве `#children` этого элемента.
2. Если этот элемент еще не был сгенерирован, вызывается функция темы по умолчанию для `#type` этого элемента. Например, если этот элемент — текстовое поле в форме (т.е. свойство `#type` установлено в `textfield` в определении формы), то функцией темы по умолчанию будет `theme_textfield()`. Если `#type` этого элемента не был установлен, по умолчанию принимается `markup`. Функции темы по умолчанию для таких основных элементов, как текстовое поле, находятся в `includes/form.inc`.
3. Если контент был сгенерирован для этого элемента, и в свойстве `#post_renderer` указано одно или более имен функций, каждая из них вызывается с передачей ей

контента и элемента. Функции `#post_renderer` должны возвращать финальный контент.

4. Присоединяет к контенту спереди `#prefix` и сзади `#suffix`, после чего возвращает его из функции.

Эффектом этой рекурсивной итерации является то, что HTML-код генерируется для каждого уровня дерева формы. Например, в форме с набором полей, содержащим два поля, элемент `#children` набора полей будет содержать HTML-код для полей внутри него, а элемент `#children` формы — весь HTML-код для формы (включая HTML-код набора полей).

Этот сгенерированный массив, готовый к визуализации, затем возвращается вызвавшей его функции `drupal_get_form()`. Это все, что требовалось. Мы добрались до финального блока “Вернуть HTML-код” на рис. 11.1.

Проверка достоверности формы

Теперь давайте вернемся к рис. 11.1, в место, где находится блок проверки “Отправлена ли форма?”. Предположим, что форма была отправлена и содержит некоторые данные, т.е. двинемся по другой ветви и рассмотрим этот случай. Механизм обработки форм Drupal определяет, была ли форма отправлена, основываясь на том, что значение `$_POST` должно быть непустым и в `$_POST['form_id']` должна присутствовать строка, соответствующая либо идентификатору формы, либо идентификатору `$form['#base']`. (См. раздел “Установка идентификатора” ранее в этой главе.) Когда соответствие найдено, Drupal производит проверку достоверности формы.

Назначение проверки достоверности — убедиться в допустимости значений, отправленных с формой. Проверка достоверности либо завершается успехом, либо терпит неудачу. Если проверка достоверности в любом месте терпит неудачу, форма будет заново отображена с показом пользователю ошибок, обнаруженных в результате проверки. Если же проверка достоверности проходит успешно, Drupal приступает к собственно обработке отправленных значений.

Проверка достоверности маркера

Первая проверка должна определить, использует ли форма механизм маркеров Drupal (см. раздел “Установка маркера” ранее в главе). Все формы Drupal, использующие маркеры, имеют уникальный маркер, который посылается с формой и, как ожидается, будет отправлен наряду с другими значениями формы. Если маркер не соответствует ожидаемому или отсутствует, проверка достоверности заканчивается неудачей (хотя остальная часть проверки все равно выполняется, чтобы могли быть выявлены и другие ошибки).

Встроенная проверка достоверности

Затем проверяются обязательные поля на предмет того, не оставлены ли они пустыми. Элементы со свойством `#maxlength` проверяются на предмет не превышения максимально допустимого количества символов. Элементы с опциями (флажки, переключатели и поля с раскрывающимся списком) просматриваются для проверки того факта, что выбранное значение действительно присутствует в исходном списке, подготовленном при построении формы.

Проверка достоверности, специфичная для элемента

Если для какого-то отдельного элемента формы определено свойство `#validate`, вызываются функции, указанные в этом свойстве, и им передаются `$form_state` и `$element`.

Обратные вызовы проверки достоверности

Наконец, идентификатор и значения формы передаются функции (функциям) проверки достоверности, которая была определена для формы (обычно ее имя состоит из идентификатора формы, дополненного суффиксом `_validate`).

Отправка формы

Если проверка достоверности завершена успешно, наступает время передать форму и ее значения функции, которая что-то выполнит в плане отправки формы. Фактически форму может обрабатывать не одна функция, поскольку свойство `#submit` может содержать массив имен функций. Каждая функция из этого массива вызывается и получает `$form` и `$form_state`.

Перенаправление пользователя

Функция, которая выполняет предварительную обработку формы, должна установить `$form_state['redirect']` в путь Drupal, по которому будет перенаправлен пользователь, такой как `node/1234`. Если в свойстве `#submit` указано много функций, для установки `$form_state['redirect']` будет использоваться возвращаемое значение только последней вызванной функции. Если ни одна из функций не устанавливает в `$form_state['redirect']` путь Drupal, пользователь возвращается на ту же самую страницу (это значение `$_GET['q']`).

Установка перенаправления в `$form_state['redirect']` функцией отправки может быть переопределено свойством `#redirect` формы:

```
$form['#redirect'] = 'node/1'
```

или так:

```
$form['#redirect'] = array('node/1', $query_string, $named_anchor).
```

Применяя термины параметров, использованные в `drupal_goto()`, последний пример можно переписать следующим образом:

```
$form['#redirect'] = array('node/1', $query, 302)
```

Определение перенаправления формы осуществляется функцией `drupal_redirect_form()` в `includes/form.inc`. Фактическое перенаправление выполнено функцией `drupal_goto()`, которая возвращает заголовок Location веб-серверу. Параметры, которые принимает `drupal_goto()`, соответствуют членам массива в последнем примере: `drupal_goto($path = '', $options = array(), $http_response_code = 302)`.

Создание базовых форм

Если вы умеете создавать собственные формы непосредственно в HTML-коде, подход Drupal может показаться поначалу несколько загадочным. Примеры, приведенные в этом разделе, нацелены на то, чтобы дать вам возможность быстро научиться создавать собственные формы. Для начала мы напишем код простого модуля, который запрашивает у пользователя имя и выводит его на экран. Мы поместим его в собственный модуль, чтобы не изменять уже существующий код. Наша форма будет иметь только два элемента — поле ввода текста и кнопку отправки. Начнем с создания файла `.info` в каталоге `sites/all/modules/custom/formexample/formexample.info` и помещения в него следующего кода:

```
name = Form Example
description = Shows how to build a Drupal form
package = Pro Drupal Development
```

```

core = 7.x
version = VERSION
files[]=formexample.module

```

Затем создадим модуль sites/all/modules/custom/formexample/formexample.module:

```

<?php
/**
 * @file
 * Использование Form API.
 */
/**
 * Реализация hook_menu().
 */
function formexample_menu() {
  $items['formexample'] = array(
    'title'           => 'View the sample form',
    'page callback'   => 'drupal_get_form',
    'page arguments'  => array('formexample_nameform'),
    'access callback' => TRUE,
    'type'            => MENU_NORMAL_ITEM
  );
  return $items;
}
/**
 * Определение формы.
 */
function formexample_nameform() {
  $form['user_name'] = array(
    '#title'           => t('Your Name'),
    '#type'            => 'textfield',
    '#description'     => t('Please enter your name.'),
  );
  $form['submit'] = array(
    '#type'            => 'submit',
    '#value'           => t('Submit')
  );
  return $form;
}
/**
 * Проверка достоверности формы.
 */
function formexample_nameform_validate($form, &$form_state) {
  if ($form_state['values']['user_name'] == 'King Kong') {
    // Уведомить Form API, что данное поле не прошло проверку.
    form_set_error('user_name',
      t('King Kong is not allowed to use this form.'));
  }
}
/**
 * Обработка отправки формы после проверки достоверности.
 */
function formexample_nameform_submit($form, &$form_state) {
  $name = $form_state['values']['user_name'];
  drupal_set_message(t('Thanks for filling out the form, %name',
    array('%name' => $name)));
}

```

Мы реализовали базовые функции, необходимые для обработки форм: одну для определения формы, еще одну для ее проверки и последнюю — для обработки отправки формы. Дополнительно был реализован хук меню, чтобы посетители могли попасть на нашу форму. Эта простая форма должна быть похожа на показанную на рис. 11.2.

Рис. 11.2. Базовая форма для ввода текста с кнопкой Submit (Отправка)

Большая часть работы состоит в начальном заполнении структуры данных формы, т.е. в описании формы для Drupal. Эта информация хранится во вложенном массиве, который описывает элементы и свойства формы и обычно содержится в переменной с именем `$form`.

Важная задача при определении формы решается в `formexample_nameform()` в предшествующем примере, где предоставляется минимальное количество информации, необходимое Drupal для отображения формы.

На заметку! В чем состоит различие между свойством и элементом? Основное отличие в том, что свойство не может иметь свойств, в то время как элементы могут. Примером свойства элемента является свойство `#type` элемента кнопки Submit (Отправка). Распознать свойство легко — оно всегда предваряется символом `#`. Мы иногда называем свойства ключами, т.к. они имеют значения, а чтобы добраться до значения, необходимо знать имя ключа. Распространенная ошибка новичков состоит в том, что они забывают поместить `#` перед именем свойства. Если вы столкнулись с сообщением об ошибке “Cannot use string offset as an array in `form.inc`” (“Невозможно использовать строковое смещение как массив в `form.inc`”), это значит, что вы, скорее всего, забыли о ведущем символе `#`.

Свойства формы

Некоторые свойства могут использоваться где угодно, а некоторые — только в определенном контексте, например, в пределах кнопки. Полный список свойств приведен в конце этой главы. Вот более сложная версия формы, чем использованная в предыдущем примере:

```
$form['#method'] = 'post';
$form['#action'] = 'http://example.com/?q=foo/bar';
$form['#attributes'] = array(
  'enctype' => 'multipart/form-data',
  'target' => 'name_of_target_frame'
);
$form['#prefix'] = '<div class="my-form-class">';
$form['#suffix'] = '</div>';
```

Значение свойства `#method` по умолчанию устанавливается в `post`, и его можно опустить. Метод `get` не поддерживается Form API и обычно не используется в Drupal, потому

что проще применять автоматический разбор аргументов пути, предлагаемый механизмом маршрутизации меню. Свойство `#action` определено в `system_element_info()` и по умолчанию принимает значение результата выполнения функции `request_uri()`. Обычно это тот же самый URL, который отображает форму.

Идентификаторы формы

Среда Drupal должна иметь некоторый способ однозначной идентификации форм, чтобы определять, какая именно форма обрабатывается, когда на странице присутствует несколько форм, и связывать формы с функциями, которые должны обрабатывать конкретную форму. Чтобы однозначно идентифицировать форму, каждой из них назначается идентификатор. Этот идентификатор определяется в вызове `drupal_get_form()`, как показано ниже:

```
drupal_get_form('mymodulename_identifier');
```

Для большинства форм идентификатор создается в соответствии с соглашением *имя модуля* плюс *идентификатор*, описывающий, что именно делает форма. Например, форма регистрации пользователя создается модулем `user` и имеет идентификатор `user_login`.

Drupal использует идентификатор формы, чтобы определить имена функций проверки достоверности, отправки и темы, принятых по умолчанию для конкретной формы. Кроме того, Drupal применяет идентификатор формы как основу для генерирования HTML-атрибута `id` в дескрипторе `<form>` для специфической формы, поэтому формы в Drupal всегда имеют уникальные идентификаторы. Вы можете переопределить этот идентификатор, установив свойство `#id`:

```
$form['#id'] = 'my-special-css-identifier';
```

Полученный в результате HTML-дескриптор будет выглядеть примерно так:

```
<form action="/path" "accept-charset="UTF-8" method="post"
  id="my-special-css-identifier">
```

Идентификатор формы также встраивается в форму в виде скрытого поля с именем `form_id`. В рассмотренном примере в качестве идентификатора формы был выбран `formexample_nameform`, потому что он описывает форму, т.е. задача нашей формы — предоставить пользователю возможность ввести свое имя. Можно было бы выбрать и вариант `formexample_form`, однако он не является достаточно описательным, к тому же впоследствии может понадобиться добавить к модулю и другие формы.

Наборы полей

Часто возникает необходимость в разделении формы на различные наборы полей; интерфейс Form API позволяет сделать это достаточно просто. Каждый набор полей (`fieldset`) определен в структуре данных и имеет поля, определенные как дочерние. Давайте добавим в пример поле для выбора предпочитаемого цвета:

```
function formexample_nameform() {
  $form['name'] = array(
    '#title' => t('Your Name'),
    '#type' => 'fieldset',
    '#description' => t('What people call you.')
  );
  $form['name']['user_name'] = array(
    '#title' => t('Your Name'),
    '#type' => 'textfield',
  );
}
```

```

    '#description' => t('Please enter your name.')
  );
  $form['color'] = array(
    '#title' => t('Color'),
    '#type' => 'fieldset',
    '#description' => t('This fieldset contains the Color field.'),
    '#collapsible' => TRUE,
    '#collapsed' => FALSE
  );
  $form['color_options'] = array(
    '#type' => 'value',
    '#value' => array(t('red'), t('green'), t('blue'))
  );
  $form['color']['favorite_color'] = array(
    '#title' => t('Favorite Color'),
    '#type' => 'select',
    '#description' => t('Please select your favorite color.'),
    '#options' => $form['color_options']['#value']
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Submit')
  );
  return $form;
}

```

Полученная в результате форма показана на рис. 11.3.

Рис. 11.3. Простая форма с набором полей

Мы использовали необязательные свойства `#collapsible` и `#collapsed`, чтобы заставить Drupal сделать второй набор полей сворачиваемым с использованием JavaScript за счет щелчка на заголовке набора полей.

Вот вопрос на сообразительность: когда `$form_state['values']` передается функциям проверки достоверности и отправки, какое значение будет иметь поле цвета — `$form_state['values']['color']['favorite_color']` или `$form_state['values']`

`['favorite_color']`? Другими словами, будет значение вложенным в набор полей или нет? Ответ: смотря когда. По умолчанию процессор формы “сглаживает” значения формы, так что следующая функция будет работать корректно:

```
function formexample_nameform_submit($form_id, $form_values) {
    $name = $form_state['values']['user_name'];
    $color_key = $form_state['values']['favorite_color'];
    $color = $form_state['values']['color_options'][$color_key];
    drupal_set_message(t('%name loves the color %color!',
        array('%name' => $name, '%color' => $color)));
}
```

Сообщение, установленное обновленным обработчиком отправки, показано на рис. 11.4.

Рис. 11.4. Сообщение от обработчика отправки формы

Однако если свойство `#tree` установлено в `TRUE`, то структура данных формы будет отражена в именах значений формы. Таким образом, если бы в объявлении формы присутствовал оператор

```
$form ['#tree'] = TRUE;
```

тогда пришлось бы обращаться к данным следующим образом:

```
function formexample_nameform_submit($form_id, $form_values) {
    $name = $form_state['values']['name']['user_name'];
    $color_key = $form_state['values']['color']['favorite_color'];
    $color = $form_state['values']['color_options'][$color_key];
    drupal_set_message(t('%name loves the color %color!',
        array('%name' => $name, '%color' => $color)));
}
```

Совет. Установка свойства `#tree` в `TRUE` дает вложенный массив полей с их значениями. Если для `#tree` указано значение `FALSE` (по умолчанию), получается сглаженное представление имен полей и значений.

Темизация форм

Drupal имеет встроенные функции, позволяющие брать структуру данных формы, которую вы определяете, и преобразовывать ее в HTML-код, т.е. визуализировать. Однако часто возникает потребность в изменении выходных данных, генерируемых системой Drupal, или в тонком управлении этим процессом. К счастью, Drupal позволяет легко делать это.

Использование `#prefix`, `#suffix` и `#markup`

Если потребности в темизации очень просты, их можно реализовать за счет использования атрибутов `#prefix` и `#suffix`, позволяющих добавить HTML-код до и/или после элементов формы:

```
$form['color'] = array(
  '#prefix' => '<hr />',
  '#title' => t('Color'),
  '#type' => 'fieldset',
  '#suffix' => '<div class="privacy-warning">' .
    t('This information will be displayed publicly!') . '</div>';
);
```

Этот код добавил бы горизонтальную линию выше набора полей Color (Цвет) и сообщение о конфиденциальности под ним, как показано на рис. 11.5.

Рис. 11.5. Свойства `#prefix` и `#suffix` добавляют контент перед и после элемента

Разметку HTML можно даже объявить как тип `#markup` в форме (хотя это используется нечасто). Любой элемент формы без свойства `#type` по умолчанию преобразуется в `#markup`.

```
$form['blinky'] = array(
  '#markup' => '<blink>Hello!</blink>'
);
```

На заметку! Считается, что этот метод ввода HTML-разметки в формы хорош настолько же, как и идея использования дескриптора `<blink>`. Он не столь безупречен, как функция темы, и обычно усложняет веб-дизайнерам работу с вашим сайтом.

Использование функции темы

Самый гибкий путь темизации форм состоит в применении функции темы к конкретной форме или ее элементу. Здесь требуется выполнить два шага. Сначала Drupal необходимо информировать о том, какие функции темы будет реализовывать модуль. Это делается посредством `hook_theme()` (подробности ищите в главе 9). Ниже показана быстрая реализация `hook_theme()` для нашего модуля, которая, по сути, говорит: “Наш модуль предоставляет две функции темы, и они могут быть вызваны без дополнительных аргументов”:

```
/**
 * Реализация hook_theme().
 */
function formexample_theme() {
  return array(
    'formexample_nameform' => array(
      'render element' => 'form',
      'template' => 'formexample-nameform',
    ),
  );
}
```

Атрибут `template` указывает, что для визуализации этой формы используется файл шаблона по имени `formexample-nameform.tpl.php`.

Следующий шаг состоит в применении функции предварительной обработки шаблона для сбора всех элементов из формы и обеспечения доступа к каждому элементу индивидуально. Это позволит темизатору управлять отображением каждого элемента на форме. Приведенная ниже функция присваивает каждый элемент формы переменной с ключом в массиве переменных, равным имени поля, т.е. `$variables['formexample_formname']['name']` — это переменная, содержащая текстовое поле, которое служит для визуализации данного поля формы.

```
/**
 * Присваивает элементы формы переменным, значения которых темизатор может
 * использовать для управления тем, как отображаются элементы формы, или
 * отображать целую форму, как было сконструировано выше.
 */
function template_preprocess_formexample_nameform(&$variables) {
  $variables['formexample_nameform'] = array();
  $hidden = array();
  // Предоставить переменные, именованные после ключей form, чтобы
  // темизатор мог выводить каждый элемент независимо.
  foreach (element_children($variables['form']) as $key) {
    $type = $variables['form'][$key]['#type'];
    if ($type == 'hidden' || $type == 'token') {
      $hidden[] = drupal_render($variables['form'][$key]);
    }
    else {
      $variables['formexample_nameform'][$key] =
        drupal_render($variables['form'][$key]);
    }
  }
  // Скрыть элементы формы, не имеющие значения для темизатора.
  $variables['formexample_nameform']['hidden'] = implode($hidden);
  // Собрать все элементы формы для упрощения их вывода как целой формы.
  $variables['formexample_nameform_form'] =
    implode($variables['formexample_nameform']);
}
```

Следующий шаг состоит в создании файла `.tpl`, который Drupal будет использовать для визуализации формы. В этом примере выводится каждое поле формы, а список для выбора цвета размещается над полем `name` за счет того, что это поле `color` формы выводится первым.

```
<?php
/**
 * @file
 *
 * Это файл шаблона для визуализации formexample_nameform.
 * Здесь каждый элемент формы визуализируется индивидуально в отличие
 * от визуализации формы целиком за раз, что обеспечивает высокую
 * степень управления компоновкой форм. Можно также вывести форму целиком
 * за раз с использованием предварительно определенной компоновки в модуле
 * путем вывода $variables['formexample_nameform_form'];
 */
print '<div id="formexample_nameform">';
print $variables['formexample_nameform']['color'];
print $variables['formexample_nameform']['name'];
print $variables['formexample_nameform']['submit'];
print $variables['formexample_nameform']['hidden'];
print '</div>';
// print $formexample_nameform_form;
?>
```

Указание Drupal, какую функцию темы использовать

Можно заставить Drupal использовать функцию, которая не соответствует схеме именования `theme_` плюс идентификатор формы, определив свойство `#theme` для формы:

```
// Теперь форма будет темизирована с помощью функции
// theme_formexample_special_theme().
$form['#theme'] = 'formexample_alternate_nameform';
```

Или же можно предложить Drupal использовать особую функцию темы только для одного элемента формы:

```
// Темизировать этот элемент набора полей с помощью
// theme_formexample_coloredfieldset().
$form['color'] = array(
  '#title' => t('Color'),
  '#type' => 'fieldset',
  '#theme' => 'formexample_coloredfieldset'
);
```

Обратите внимание, что в обоих случаях определяемая в свойстве `#theme` функция должна быть известна реестру тем, т.е. она должна быть объявлена где-то в реализации `hook_theme()`.

На заметку! Drupal будет предварять строку, которую вы указываете в `#theme`, префиксом `theme_`. Таким образом, `#theme` устанавливается в `formexample_coloredfieldset`, а не в `theme_formexample_coloredfieldset`, даже при том, что имя функции темы, которую будут вызывать, стоит последним. За детальной информацией обращайтесь в главу 9.

Спецификация функций проверки достоверности и отправки формы с помощью `hook_forms()`

Иногда бывают особые случаи, когда необходимо иметь множество различных форм, но при этом использовать только одну функцию проверки достоверности или отправки формы. Это называется *многократным использованием кода*, и оно является хорошим решением в ситуациях подобного рода. Модуль ноды, например, прогоняет все разновидности типов нод через свои функции проверки достоверности и отправки. Таким образом, нужен способ отображения множества идентификаторов форм на функции проверки достоверности и обработки данных. Это делается в хуке `hook_forms()`.

Когда среда Drupal извлекает форму, она сначала ищет функцию, определяющую форму, основываясь на идентификаторе формы (в нашем коде для этой цели служила функция `formexample_nameform()`). Если Drupal не находит такой функции, она вызывает хук `hook_forms()`, который делает запрос ко всем модулям с предложением отобразить идентификаторы форм на обратные вызовы. Например, `node.module` использует следующий код, чтобы отобразить все различные виды идентификаторов нод формы на один хук:

```
/**
 * Реализация hook_forms(). Все формы нод разделяют
 * один и тот же обработчик формы.
 */
function node_forms() {
  $forms = array();
  if ($types = node_get_types()) {
    foreach (array_keys($types) as $type) {
      $forms[$type . '_node_form']['callback'] = 'node_form';
    }
  }
  return $forms;
}
```

В нашем примере формы можно было бы реализовать `hook_forms()`, чтобы отобразить другой идентификатор формы на существующий код:

```
/**
 * Реализация hook_forms().
 */
function formexample_forms($form_id, $args) {
  $forms['formexample_special'] = array(
    'callback' => 'formexample_nameform');
  return $forms;
}
```

Если теперь вызвать `drupal_get_form('formexample_special')`, то Drupal сначала проверит, существует ли функция по имени `formexample_special()`, определяющая форму. Если эта функция не найдена, будет вызван хук `hook_forms()`, и Drupal обнаружит, что идентификатор формы `formexample_special` отображен на `formexample_nameform`. Drupal вызовет `formexample_nameform()`, чтобы получить определение формы, а затем попытается вызвать `formexample_special_validate()` и `formexample_special_submit()`, соответственно, для проверки достоверности и отправки формы.

Порядок вызова функций тем, проверки достоверности и отправки

Как видите, существует несколько мест, в которых можно дать системе Drupal информацию о местонахождении функций темы, проверки достоверности и отправки формы. Наличие множества вариантов может запутать, поэтому ниже приводится общий обзор тех мест, где Drupal ищет соответствующую функцию темы. При этом предполагается, что используется тема с именем `mytheme`, и осуществляется вызов `drupal_get_form('formexample_nameform')`. Это, однако, зависит от вашей реализации `hook_theme()`.

Если `$form['#theme']` установлено в `'foo'` в определении формы, то последовательность проверок, выполняемых Drupal, выглядит следующим образом.

1. `themes/mytheme/foo.tpl.php` // Шаблонный файл, предоставленный темой.
2. `formexample/foo.tpl.php` // Шаблонный файл, предоставленный модулем.
3. `mytheme_foo()` // Функция, предоставленная темой.
4. `phptemplate_foo()` // Функция темы, предоставленная механизмом тем.
5. `theme_foo()` // `'theme_'` плюс значение `$form['#theme']`.

Однако если `$form['#theme']` не установлено в определении формы, то последовательность будет такой.

1. `themes/mytheme/formexample-nameform.tpl.php` // Шаблон, предоставленный темой.
2. `formexample/formexample-nameform.tpl.php` // Шаблонный файл,
// предоставленный модулем.
3. `mytheme_formexample_nameform()` // Функция темы, предоставленная темой.
4. `phptemplate_formexample_nameform()` // Функция темы, предостав-
// ленная механизмом тем.
5. `theme_formexample_nameform()` // `'theme_'` плюс идентификатор формы.

Во время проверки достоверности формы функция проверки устанавливается в следующем порядке.

1. Функция, определенная `$form['#validate']`
2. `formexample_nameform_validate` // Идентификатор формы плюс `'_validate'`

Функция для обработки отправки формы определяется в таком порядке.

1. Функция, определенная `$form['#submit']`
2. `formexample_nameform_submit` // Идентификатор формы плюс `'_submit'`

Помните, что формы могут иметь множество функций проверки и отправки.

Реализация функции проверки достоверности

Drupal имеет встроенный механизм для подсветки элементов формы, не прошедших проверку достоверности, и отображения пользователю сообщений об ошибках. Чтобы понять, как этот механизм работает, рассмотрим функцию проверки достоверности в нашем примере:

```
/**
 * Проверка достоверности формы.
 */
function formexample_nameform_validate($form_id, $form_state) {
  if ($form_state['values']['user_name'] == 'King Kong') {
    // Уведомить Form API, что данное поле не прошло проверку.
    form_set_error('user_name',
      t('King Kong is not allowed to use this form.'));
  }
}
```

Обратите внимание на использование `form_set_error()`. Когда King Kong открывает форму и вводит свое имя, он видит в верхней части окна сообщение об ошибке, а содержимое поля, которое содержит ошибку, подсвечивается красным (рис. 11.6).

Рис. 11.6. Отрицательные результаты проверки достоверности показываются пользователю

Возможно, ему следовало бы использовать Kong в качестве имени. Так или иначе, дело в том, что функция `form_set_error()` обнаружила ошибку в форме, поэтому проверка достоверности завершилась неудачей.

Функции проверки достоверности должны осуществлять именно проверку. Как правило, они не должны изменять данные. Тем не менее, они могут добавлять информацию в массив `$form_values`, как показано в следующем разделе. Если функция проверки достоверности выполняет большой объем вычислений, и вы хотите сохранить полученный результат, чтобы использовать его в функции отправки формы, доступны два выбора. Можно использовать `form_set_value()` или же `$form_state`.

Использование для передачи данных `form_set_value()`

Наиболее формальный выбор состоит в создании элемента формы для упаковки данных при написании функции определения формы и последующем использовании `form_set_value()` для сохранения данных. Сначала потребуется создать заполнитель для элемента формы:

```
$form['my_placeholder'] = array(
  '#type' => 'value',
  '#value' => array()
);
```

Затем, во время работы функции проверки достоверности, данные сохраняются:

```
// Работа по генерации $my_data как часть проверки достоверности.
...
// Сохранение результатов работы.
form_set_value($form['my_placeholder'], $my_data, &$form_state);
```

Теперь можно обратиться к данным в функции отправки:

```
// Вместо повторения работы, которая делалась в функции проверки
// достоверности, можно просто воспользоваться сохраненными данными.
$my_data = $form_state['values']['my_placeholder'];
```

Предположим, что нужно преобразовать данные к стандартному представлению. Например, в базе данных имеется список кодов стран, на основе которого производится проверка, но ваш неблагоразумный босс настаивает, чтобы пользователи могли вводить названия стран в текстовых полях. В этом случае потребовалось бы создать в форме заполнитель и проверять введенный пользователем текст, используя разнообразные трюки для того, чтобы распознать, скажем, “The Netherlands” и “Nederland” как Нидерланды, для которых в соответствии со стандартом ISO 3166 предусмотрен код “NL”.

```
$form['country'] = array(
  '#title' => t('Country'),
  '#type' => 'textfield',
  '#description' => t('Enter your country.')
);

// Создать заполнитель. Он будет заполняться
// во время проверки достоверности.
$form['country_code'] = array(
  '#type' => 'value',
  '#value' => ''
);
```

В функции проверки достоверности понадобилось бы сохранить код страны в заполнителе:

```
// Выяснить, имеется ли соответствие.
$country_code = formexample_find_country_code($form_state['values']['country']);
if ($country_code) {
  // Соответствие найдено. Сохраняем его так,
  // чтобы обработчик отправки мог видеть его.
  form_set_value($form['country_code'], $country_code, &$amp;$form_state);
}
else {
  form_set_error('country', t('Your country was not recognized. Please use
  a standard name or country code.'));
}
```

Теперь обработчик отправки формы может обратиться к коду страны в `$form_values['country_code']`.

Использование для передачи данных `$form_state`

Более простой подход состоит в применении для хранения значения `$form_state`. Поскольку `$form_state` передается и в функцию проверки достоверности, и в функцию отправки по ссылке, функция проверки может сохранить данные для того, чтобы функция отправки их затем увидела. Использование пространства имен вашего модуля внутри `$form_state` вместо создания ключа — очень неплохая идея.

```
// Работа по генерации $weather_data от медленной
// веб-службы как часть процесса проверки достоверности.
...
// Сохранить результат работы в $form_state.
$form_state['mymodulename']['weather'] = $weather_data
```


Затем можно обратиться к этим данным в функции отправки:

```
// Вместо повторения работы, выполненной в функции проверки,
// можно просто воспользоваться сохраненными данными.
$weather_data = $form_state['mymodulename']['weather'];
```

Вы можете спросить: “А почему не сохранить значение в `$form_state['values']` наряду с прочими значениями полей формы?”. Это также будет работать, но имейте в виду, что `$form_state['values']` является местом для значений полей формы, а не случайных данных, сохраняемых модулями. Поскольку Drupal позволяет любому модулю присоединить функции проверки и отправки к любой форме, вы не можете исходить из предположения, что ваш модуль будет единственным работающим с состоянием формы (`$form_state`), и потому данные должны быть сохранены в согласованном и предсказуемом виде.

Проверка достоверности, специфичная для элемента

Как правило, для формы используется одна функция проверки достоверности. Однако можно обеспечить проверку и для отдельных элементов формы. Чтобы сделать это, присвойте свойству `#element_validate` элемента массив с именами функций проверки достоверности. Полная копия ветви элемента структуры данных формы будет передаваться в первом параметре. Ниже приведен несколько надуманный пример, в котором мы вынуждаем пользователя ввести в текстовое поле слово `spicy` или `sweet`:

```
// Сохранить допустимые варианты в определении формы.
$allowed_flavors = array(t('spicy'), t('sweet'));
$form['flavor'] = array(
  '#type' => 'textfield',
  '#title' => 'flavor',
  '#allowed_flavors' => $allowed_flavors,
  '#element_validate' => array('formexample_flavor_validate')
);
```

Функция проверки достоверности элемента выглядит следующим образом:

```
function formexample_flavor_validate($element, $form_state) {
  if (!in_array($form_state['values']['flavor'], $element['#allowed_flavors']))
  {
    form_error($element, t('You must enter spicy or sweet.'));
  }
}
```

Функция проверки достоверности формы будет по-прежнему вызываться после вызова всех функций проверки достоверности для элементов.

Совет. Используйте `form_set_error()`, когда есть имя элемента формы, не прошедшего проверку, и `form_error()`, когда имеется сам элемент. Последняя функция — это просто оболочка для предыдущей.

Перестройка формы

Во время проверки достоверности формы вы можете решить, что информации, поступившей от пользователя, недостаточно. Например, пропустив значения формы через механизм анализа текста, вы определяете, что введенный пользователем контент с высокой вероятностью является спамом. В результате понадобится отобразить форму снова (заполненную значениями, введенными пользователем), но добавить CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart —

Полностью автоматизированный открытый тест Тьюринга по распознаванию людей и машин; выглядит как графическое изображение с искаженными цифрами и/или буквами; иногда это называют “капча”), чтобы выяснить, что пользователь — не робот. О необходимости перестройки формы можно просигнализировать Drupal, установив `$form_state['rebuild']` внутри функции проверки — примерно так:

```
$spam_score = spamservice($form_state['values']['my_textarea']);
if ($spam_score > 70) {
  $form_state['rebuild'] = TRUE;
  $form_state['formexample']['spam_score'] = $spam_score;
}
```

В функции определения формы понадобится приблизительно такой код:

```
function formexample_nameform($form_state) {

  // Нормальное определение формы.
  ...

  if (isset($form_state['formexample']['spam_score'])) {
    // Если это установлено, перестроить форму;
    // добавить к форме элемент captcha.
    ...
  }
  ...
}
```

Реализация функции отправки

Функция отправки — это функция, которая обеспечивает фактическую обработку формы после того, как последняя была проверена на предмет достоверности данных. Она выполняется только в случае, если проверка достоверности формы прошла успешно, и форма не помечена как подлежащая перестройке. Предполагается, что функция отправки модифицирует `$form_state['redirect']`.

```
function formexample_form_submit($form, $form_state) {

  // Выполнение каких-то действий.
  ...

  // Направление пользователя к ноде номер 3.
  $form_state['redirect'] = 'node/3';
}
```

Если имеется несколько функций, обрабатывающих отправку формы (см. раздел “Отправка формы” ранее в этой главе), последнее слово останется за последней функцией, установившей `$form_state['redirect']`. Перенаправление функции отправки может быть переопределено с помощью свойства `#redirect` в форме (см. раздел “Перенаправление пользователя” ранее в этой главе). Это часто делается с помощью хука `hook_form_alter()`.

Совет. Флаг `$form_state['rebuild']` может быть установлен также и в функциях отправки, как это делалось в функциях проверки достоверности. Если он установлен, то все функции отправки будут выполнены, но любые значения перенаправления будут игнорироваться, и форма будет перестроена с использованием отправленных значений. Это может оказаться полезным для добавления к форме дополнительных полей.

Изменение форм с помощью `hook_form_alter()`

Используя хук `hook_form_alter()`, можно изменить любую форму. Все, что требуется для этого знать — идентификатор формы. Существуют два подхода к изменению форм.

Изменение любой формы

Давайте изменим регистрационную форму, которая показывается в блоке входа в систему пользователя и на странице входа пользователя в систему:

```
function formexample_form_alter(&$form, &$form_state, $form_id) {
  // Этот код вызывается при каждом построении формы Drupal;
  // оператор if позволяет реагировать только на блок
  // входа и форму входа пользователя.
  if ($form_id == 'user_login_block' || $form_id == 'user_login') {
    // Добавляем предупреждение в форму входа пользователя.
    $form['warning'] = array(
      '#markup' => t('We log all login attempts!'),
      '#weight' => -5
    );

    // Изменить 'Log in' на 'Sign in'.
    $form['submit']['#value'] = t('Sign in');
  }
}
```

Поскольку `$form` передается по ссылке, мы имеем здесь полный доступ к определению формы и можем вносить в него любые изменения. В данном примере добавлен некоторый текст с использованием элемента формы по умолчанию (см. раздел “Элемент markup” далее в этой главе), а также изменено значение кнопки отправки формы.

Изменение определенной формы

Предыдущий подход работает, но поскольку множество модулей изменяют формы и каждая форма передается каждой реализации `hook_form_alter()`, у вас в голове должен прозвучать сигнал тревоги. “Это же дорого!” — возможно, подумаете вы. “Почему бы просто не сконструировать функцию из идентификатора формы и не вызвать ее?”. Вы на правильном пути. Именно так поступает Drupal. Так что следующая функция также изменит форму входа пользователя:

```
function formexample_form_user_login_alter(&$form, &$form_state) {
  $form['warning'] = array(
    '#value' => t('We log all login attempts!'),
    '#weight' => -5
  );

  // Изменить 'Log in' на 'Sign in'.
  $form['submit']['#value'] = t('Sign in');
}
```

Имя функции конструируется из следующих ингредиентов:

имя_модуля + 'form' + идентификатор_формы + 'alter'

Например, для

'formexample' + 'form' + 'user_login' + 'alter'

получается такое имя:

formexample_form_user_login_alter

В этом конкретном случае предпочтение отдается первой форме `hook_form_alter()`, потому что участвуют два идентификатора формы (`user_login` для формы `http://example.com/?q=user` и `user_login_block` — для формы, появляющейся в блоке `user`).

Программная отправка формы с помощью `drupal_form_submit()`

Любая форма, которая отображается в веб-браузере, может быть также заполнена программно. Давайте заполним поля с именем (`user_name`) и предпочитаемым цветом (`favorite_color`) программно:

```
$form_id = 'formexample_nameform';
$field_state['values'] = array(
  'user_name' => t('Marvin'),
  'favorite_color' => t('green')
);
// Отправить форму, используя эти значения.
drupal_form_submit($form_id, $form_state);
```

Вот и все, что необходимо. Достаточно лишь вызвать `drupal_form_submit()`, передав ей идентификатор формы и значения для формы.

На заметку! Многие функции отправки предполагают, что пользователь, делающий запрос — это пользователь, отправляющий форму. При программной отправке формы вы не должны забывать об этом, поскольку упомянутые пользователи не обязательно совпадают.

Динамические формы

До сих пор мы рассматривали простые одностраничные формы. Но, возможно, понадобится сделать так, чтобы пользователи заполняли форму, которая динамически отображает элементы на основе уже заполненных других элементов формы. В следующем примере показано, как динамически отображать элементы по мере выбора пользователем различных опций на форме.

Для начала необходимо создать в папке `site/all/modules/custom` каталог по имени `form_example_dynamic`. Затем в этом каталоге нужно создать файл `form_example_dynamic.info` со следующим содержанием:

```
name = Form Example — Creating a Dynamic Form
description = An example of a dynamic form.
package = Pro Drupal Development
core = 7.x
files[] = form_example_dynamic.module
```

Далее понадобится создать файл `form_example_dynamic.module` и поместить в него следующий заголовок:

```
<?php
/**
 * @file
 * Пример использования нового элемента #states из Form API, который
 * обеспечивает динамическое поведение форм с очень простой настройкой.
 */
```

Имея такой заголовок, далее необходимо создать элемент меню, с помощью которого посетители смогут получать доступ к новой форме. Модуль предоставляет одиночный элемент меню, который может быть доступен через `www.example.com/form_example_dynamic`.

```

/**
 * Реализация hook_menu().
 */
function form_example_dynamic_menu() {
  $items['form_example_dynamic'] = array(
    'title' => t('Form Example Dynamic Form'),
    'page callback' => 'drupal_get_form',
    'page arguments' => array('form_example_dynamic_form'),
    'access callback' => TRUE,
    'type' => MENU_NORMAL_ITEM
  );
  return $items;
}

```

Теперь, когда построение меню завершено, можно приступить к созданию формы. Первый элемент, отображаемый на форме, представляет собой последовательность из трех переключателей, которые позволяют посетителю сайта указывать тип арендуемого помещения:

```

function form_example_dynamic_form($form, &$form_state) {
  $form['room_type'] = array(
    '#type' => 'radios',
    '#options' => drupal_map_assoc(array(t('Study Room'),
    t('Small Conference Room'), t('Board Room'))),
    '#title' => t('What type of room do you require?')
  );
}

```

Следующий элемент формы — это набор полей, которые содержат детали о помещении и используют атрибут `#states` для определения, когда этот элемент должен отображаться на странице. Атрибут `#states` устанавливает признак видимости этого набора полей, проверяя переключатели `room_type` на предмет того, выбран ли вариант Study Room (Кабинет). Если переключатель Study Room выбран, значение устанавливается в `true`, и форма будет визуализировать набор полей с применением jQuery. Синтаксис проверки видимости соответствует синтаксису использования селекторов в jQuery. В нашем случае просматривается элемент ввода (переключатели) по имени `room_type`. Мы проверяем, равно ли значение элемента ввода Study Room.

```

$form['study_room'] = array(
  '#type' => 'fieldset',
  '#title' => t('Study Room Details'),
  '#states' => array(
    'visible' => array(
      ':input[name="room_type"]' => array('value' => t('Study Room')),
    ),
  ),
);

```

Следующими элементами, отображаемыми на форме, являются два флажка, которые позволяют посетителю предоставить детали об оснащении для установки в кабинете. В рассматриваемом примере выбор ограничивается только стульями и ПК. Применяется тот же самый подход с атрибутом `#states`, что и для предыдущего набора полей. Флажки должны отображаться, только если посетитель выбрал вариант Study Room.

```

$form['study_room']['equipment'] = array(
  '#type' => 'checkboxes',
  '#options' => drupal_map_assoc(array(t('Chairs'), t('PC'))),
  '#title' => t('What equipment do you need?'),
);

```

```

'#states' => array(
  'visible' => array( // Выполняемое действие.
    ':input[name="room_type"]' => array('value' => t('Study Room')),
  ),
),
);

```

Если пользователь отметил флажок Chairs (Стулья), должно отобразиться текстовое поле, позволяющее ввести количество стульев, которые необходимо установить в кабинете до прихода посетителей. Видимость этого текстового поля управляется с помощью #states; поле отображается, только если пользователь отметил флажок Chairs.

```

$form['study_room']['chairs'] = array(
  '#type' => 'textfield',
  '#title' => t('How Many Chairs Do You Need?:'),
  '#size' => 4,
  '#states' => array(
    'visible' => array( // Выполняемое действие.
      ':input[name="equipment[Chairs]"'] => array('checked' => TRUE),
    ),
  ),
);

```

Следующим элементом формы является еще одно текстовое поле, позволяющее посетителю вводить детали о типе ПК, который должен быть установлен в кабинете. Как и для элемента, отвечающего за стулья, видимость данного элемента управляется с использованием #states путем проверки, отметил ли посетитель флажок PC (ПК).

```

$form['study_room']['pc'] = array(
  '#type' => 'textfield',
  '#title' => t('What Type of PC do you need?:'),
  '#size' => 15,
  '#states' => array(
    'visible' => array( // Выполняемое действие.
      ':input[name="equipment[PC]"'] => array('checked' => TRUE),
    ),
  ),
);

```

Следующий набор элементов формы отображается, только в случае выбора посетителем переключателя Small Conference Room (Небольшой конференц-зал). При этом для определения видимости элементов формы также применяется тот же самый шаблон с атрибутом #states.

```

$form['small_conference_room'] = array(
  '#type' => 'fieldset',
  '#title' => t('small_conference_room Information'),
  '#states' => array(
    'visible' => array(
      ':input[name="room_type"]' => array('value' => t('Small Conference Room')),
    ),
  ),
);

$form['small_conference_room']['how_many_pcs'] = array(
  '#type' => 'select',
  '#title' => t('How many PCs do you need set up in the small conference room?'),
  // Сколько ПК нужно установить в небольшом конференц-зале?
);

```

```

    '#options' => array(
      1 => t('One'),
      2 => t('Two'),
      3 => t('Three'),
      4 => t('Four'),
      5 => t('Lots'),
    ),
  );

$form['small_conference_room']['comment'] = array(
  '#type' => 'item',
  '#description' => t("Wow, that's a long time."), // Слишком долго
  '#states' => array(
    'visible' => array(
      ':input[name="how_many_pcs"]' => array('value' => '5'),
    ),
  ),
);

$form['small_conference_room']['room_name'] = array(
  '#type' => 'textfield',
  '#title' => t('Which room do you want to use?:'),
  // Какое помещение планируется использовать
);

$form['small_conference_room']['hours'] = array(
  '#type' => 'select',
  '#options' => drupal_map_assoc(array(t('Free'), t('Paid'))),
  '#title' => t('Do you want to reserve the room when it is free (no fees) or
paid (prime time)?'),
  // Каким образом резервировать помещение:
  // когда оно свободно (без оплаты) или
  // на платной основе (основное время)
);

```

Следующий элемент формы выполняет две условных проверки для определения, должно ли отображаться текстовое поле. С помощью действия можно перечислить любое количество условий, которые должны быть удовлетворены перед тем, как элемент формы будет отображен. В этом случае производится проверка, какой вариант выбрал посетитель в поле Do you want to reserve the room when it is free (no fees) or paid (prime time)? (Каким образом резервировать помещение: когда оно свободно (без оплаты) или на платной основе (основное время)) — Free (Бесплатно) или Paid (На платной основе).

```

$form['small_conference_room']['hours_writein'] = array(
  '#type' => 'textfield',
  '#size' => 50,
  '#title' => t('Please enter the date and time you would like to reserve
the room and the duration.'),
  // Ввести дату, время и продолжительность аренды помещения
  '#states' => array(
    'visible' => array( // Выполняемое действие: сделать видимым.
      ':input[name="hours"]' => array('value' => t('Free')),
      ':input[name="hours"]' => array('value' => t('Paid')),
    ),
  ),
);

```

В элементе `reminder` формы демонстрируется новая проверка видимости, предусматривающая верификацию того, что посетитель выбрал `Free` или `Paid` и ввел что-либо в поле `hours_writein`.

```
$form['small_conference_room']['reminder'] = array(
  '#type' => 'item',
  '#description' => t('Remember to enter the date, start time, and end time.'),
  '#states' => array(
    'visible' => array(
      'input[name="hours"]' => array('value' => t('Free')),
      'input[name="hours"]' => array('value' => t('Paid')),
      'input[name="hours_writein"]' => array('filled' => TRUE),
    ),
  ),
);

$form['board_room'] = array(
  '#type' => 'fieldset',
  '#title' => t('Board Room Information'),
  '#states' => array(
    'visible' => array(
      ':input[name="room_type"]' => array('value' => t('Board Room')),
    ),
  ),
);

$form['board_room']['more_info'] = array(
  '#type' => 'textarea',
  '#title' => t('Please enter the date and time of when you would like to
reserve the board room'),
);

$form['board_room']['info_provide'] = array(
  '#type' => 'checkbox',
  '#title' => t('Check here if you have provided information above'),
  '#disabled' => TRUE,
  '#states' => array(
    'checked' => array( // Action to take: check the checkbox.
      ':input[name="more_info"]' => array('filled' => TRUE),
    ),
  ),
);

$form['expand_more_info'] = array(
  '#type' => 'checkbox',
  '#title' => t('Check here if you want to add special instructions.'),
);

$form['more_info'] = array(
  '#type' => 'fieldset',
  '#title' => t('Special Instructions'),
  '#collapsible' => TRUE,
  '#collapsed' => TRUE,
  '#states' => array(
    'expanded' => array(
      ':input[name="expand_more_info"]' => array('checked' => TRUE),
    ),
  ),
);
```



```

$form['more_info']['feedback'] = array(
  '#type' => 'textarea',
  '#title' => t('Please provide any additional details that will help us
better serve you.'),
);

$form['submit'] = array(
  '#type' => 'submit',
  '#value' => t('Submit your information'),
);

return $form;
}

function form_example_dynamic_form_submit($form, &$form_state) {
  drupal_set_message(t('Submitting values: @values', array('@values' =>
var_export($form_state['values'], TRUE))));
}

```

После завершения написания модель необходимо активировать и посетить форму по адресу www.example.com/form_example_dynamic. Первая страница формы должна выглядеть подобно показанной на рис. 11.7.

The screenshot shows a web form titled "Form Example Dynamic Form". Below the title is the question "What type of room do you require?". There are three radio button options: "Study Room", "Small Conference Room", and "Board Room". Below these is a checkbox labeled "Check here if you want to add special instructions.". Under the checkbox is a text input field with a placeholder "Special Instructions". At the bottom of the form is a "Submit your information" button.

Рис. 11.7. Начальное состояние формы

Выбор переключателя Study Room (Кабинет) приводит к открытию следующей части формы (рис. 11.8), где у посетителя запрашивается оснащение, которое должно быть установлено в помещении.

Выбор переключателя Small Conference Room (Небольшой конференц-зал) приводит к отображению элементов формы, относящихся к конференц-залу (рис. 11.9).

Если посетитель выбирает переключатель Board Room (Зал заседаний), отображаются элементы, показанные на рис. 11.10.

С использованием только что описанного подхода можно создавать все многообразие форм, предназначенных для организации онлайн-опросов.

Form Example Dynamic Form

What type of room do you require?

☒ Study Room

☐ Small Conference Room

☐ Board Room

Study Room Details

What equipment do you need?

☐ Chairs

☐ PC

☐ Check here if you want to add special instructions.

— ▷ Special Instructions —

Submit your information

Рис. 11.8. Набор полей Study Room Details (Детали о кабинете) отображается на основе ранее выбранного переключателя

Form Example Dynamic Form

What type of room do you require?

☐ Study Room

☒ Small Conference Room

☐ Board Room

small_conference_room Information

How many PCs do you need set up in the small conference room?

One

Which room do you want to use?:

Do you want to reserve the room when it is free (no fees) or paid (prime time)?

Free

☐ Check here if you want to add special instructions.

— ▷ Special Instructions —

Submit your information

Рис. 11.9. После выбора переключателя Small Conference Room отображаются элементы формы, относящиеся к конференц-залу

Form Example Dynamic Form

What type of room do you require?

☐ Study Room
☐ Small Conference Room
☒ Board Room

Board Room Information

Please enter the date and time of when you would like to reserve the board room

☐ Check here if you have provided information above

☐ Check here if you want to add special instructions.

Special Instructions

Рис. 11.10. После выбора переключателя Board Room отображаются элементы, относящиеся к залу заседаний

Свойства Form API

При создании определения формы в функции построения формы, для представления информации о форме используются ключи в виде массивов. Наиболее общие ключи перечислены в следующих разделах. Некоторые ключи добавляются автоматически строителем форм.

Свойства для корневого элемента формы

Свойства, описанные в следующих разделах, специфичны для корневого элемента формы. Другими словами, можно установить `$form['#programmed'] = TRUE`, но установка, скажем, `$form['набор_полей']['текстовое_поле']['#programmed'] = TRUE` не будет иметь смысла для строителя форм.

Свойство `#action`

Путь, по которому форма будет отправлена.

Свойство `#built`

Используется для выяснения, построен ли уже элемент формы.

Свойство `#method`

Метод HTTP, с помощью которого форма будет отправлена.

Свойство `#redirect`

Указывает, куда переходить по умолчанию после того, как форма отправлена. Это значение должно возвращаться функцией обратного вызова отправки формы. Изменение значения `#redirect` другой формы в хуке `hook_form_alter()` позволяет изменить место перенаправления формы после ее отправки.

Свойства, добавляемые ко всем элементам

Когда построитель формы проходит по определению формы, он удостоверяется, что каждый элемент имеет значение по умолчанию. Значения по умолчанию устанавливаются в функции `_element_info()` внутри `includes/form.inc`, но могут быть переопределены определением элемента в `hook_elements()`.

Свойство `#description`

Это строковое свойство добавляется ко всем элементам и по умолчанию ему присваивается значение `NULL`. Оно визуализируется функцией темы элемента. Например, как показано на рис. 11.2, описание текстового поля отображается под этим текстовым полем.

Свойство `#attributes`

С использованием этого свойства можно устанавливать дополнительные атрибуты HTML, такие как `class`. В следующем примере класс CSS формы устанавливается в `search-form`:

```
<?php
$form['#attributes'] = array('class' => 'search-form');
?>
```

Свойство `#required`

Это булевское свойство добавляется ко всем элементам и по умолчанию ему присваивается значение `FALSE`. Изменение этого значения на `TRUE` приведет к тому, что встроенная система проверки достоверности формы Drupal будет выдавать сообщение об ошибке, если форма отправлена, но поле не заполнено. Кроме того, если указано значение `TRUE`, для этого элемента будет установлен класс CSS (см. `theme_form_element()` в `includes/form.inc`).

Свойство `#tree`

Это булевское свойство добавляется ко всем элементам и по умолчанию имеет значение `FALSE`. Если установить его в `TRUE`, массив `$form_values`, получающийся в результате отправки формы, не будет сглажен. Это влияет на способ доступа к отправленным значениям (см. раздел “Наборы полей” ранее в этой главе).

Свойства, доступные во всех элементах

Свойства, рассмотренные в последующих разделах, доступны во всех элементах.

Свойство `#type`

Эта строка объявляет тип элемента. Например, `#type = 'textfield'`. Корневой элемент формы должен иметь объявление `#type = 'form'`.

Свойство `#access`

Это булевское свойство определяет, показывается ли элемент пользователю. Если элемент имеет дочерние элементы, последние не будут видны, если свойство `#access` родительского элемента установлено в `FALSE`. Например, если элемент является набором полей, то ни одно из полей, включенных в этот набор, не будет показано, если значением `#access` набора является `FALSE`.

Свойство `#access` может быть установлено в `TRUE` или `FALSE` напрямую или же с помощью функции, которая возвращает `TRUE` или `FALSE` в качестве своего результата. Функция будет выполнена, когда извлекается определение формы. Вот пример формы ноды Drupal по умолчанию:

```
$form['revision_information']['revision'] = array(
  '#access' => user_access('administer nodes'),
  '#type' => 'checkbox',
  '#title' => t('Create new revision'),
  '#default_value' => $node->revision,
);
```

Свойство #after_build

Массив имен функций, которые будут вызываться после построения формы или элемента.

Свойство #array_parents

Массив имен родителей элемента (включая его самого) в форме. Он будет всегда соответствовать структуре \$form. Это отличается от #parents только структурой, используемой в \$form_state['values'], которая является плоской, если свойство #tree не установлено в TRUE.

Свойство #attached

Массив с ключами, содержащий пары тип => значение, где тип (чаще всего css, js и library) определяет технику загрузки, а значение задает опции, передаваемые функции загрузчика.

Свойство #default_value

Это свойство имеет смешанный тип. Для элементов ввода оно представляет собой значение, которое используется в поле, если форма пока еще не отправлена. Не путайте его с элементом #value, определяющим внутреннее значение формы, которое никогда не передается пользователю и доступно в \$form_state['values'].

Свойство #disabled

Отключает (делает недоступным) элемент ввода формы. Обратите внимание, что отключение поля формы не обязательно предотвращает изменение его значения через манипуляцию деревом DOM. Это просто сообщает браузеру о запрете ввода в это поле.

Свойство #element_validate

Список специальных функций проверки достоверности, которые должны быть переданы.

Свойство #parents

Это свойство типа массива добавляется ко всем элементам, и его значение по умолчанию — пустой массив. Оно используется внутренне строителем формы, чтобы идентифицировать родительские элементы дерева формы. За дополнительной информацией обращайтесь по адресу <http://drupal.org/node/48643>.

Свойство #post_render

Функция (или функции) для вызова после выполнения визуализации в drupal_render(). Указанная функция вызывается с двумя аргументами — визуализированным элементом и его дочерним элементом. Она возвращает (потенциально) измененное содержимое элемента.

Свойство #prefix

Строка, определенная в этом свойстве, будет добавлена к выходным данным при визуализации элемента, непосредственно перед визуализируемым элементом.

Свойство `#pre_render`

Функция (или функции) для вызова перед выполнением визуализации в `drupal_render()`. Указанная функция (функции) получает элемент в качестве аргумента и должна возвращать измененный элемент.

Свойство `#process`

Это свойство представляет собой ассоциативный массив. Каждый элемент массива состоит из имени функции в виде ключа и параметров, передаваемых функции, в виде значений. Эти функции вызываются во время построения элемента и позволяют проводить с ним дополнительные манипуляции во время построения формы. Например, в модуле `modules/system/system.module`, в котором определен тип `checkboxes`, функция `expand_checkboxes()` в `includes/form.inc` устанавливается так, чтобы она могла вызываться в ходе построения формы:

```
$type['checkboxes'] = array(
  '#input' => TRUE,
  '#process' => array('expand_checkboxes'),
  '#tree' => TRUE
);
```

Свойство `#states`

Добавляет к элементу код JavaScript, чтобы позволить ему иметь различные активные состояния.

Свойство `#suffix`

Строка, определенная в этом свойстве, будет добавлена к выходным данным при визуализации элемента, сразу после визуализируемого элемента.

Свойство `#theme`

Это необязательное свойство определяет строку, которая будет использоваться, когда Drupal ищет функцию темы для данного элемента. Например, установка `#theme = 'foo'` заставит Drupal проверить реестр тем на предмет записи, соответствующей `foo`. См. раздел “Обнаружение функции темы для формы” ранее в этой главе.

Свойство `#theme_wrappers`

Функция темы, которая будет вызываться для элемента после того, как элемент и его дочерние элементы визуализированы, но перед тем, как будут вызваны функции `#post_render`.

Свойство `#title`

Эта строка представляет собой заголовок элемента.

Свойство `#tree`

Используется для поддержки коллекций элементов формы. Обычно применяется к “родительскому” элементу, т.к. свойство `#tree` каскадирует подэлементы.

Свойство `#weight`

Это свойство может иметь в качестве значения целое или десятичное число, задающее вес элемента. Когда элементы формы визуализируются, они сортируются в соответствии с их весами. Те, у которых меньший вес, “всплывают” и окажутся выше; элементы с большим весом “опускаются” и появятся ниже на визуализированной странице.

Элементы формы

В этом разделе будут представлены примеры встроенных элементов формы Drupal.

Элемент *textfield*

Пример элемента *textfield* приведен ниже:

```
$form['pet_name'] = array(
  '#title' => t('Name'),
  '#type' => 'textfield',
  '#description' => t('Enter the name of your pet.'),
  '#default_value' => $user->pet_name,
  '#maxlength' => 32,
  '#required' => TRUE,
  '#size' => 15,
  '#weight' => 5,
  '#autocomplete_path' => 'pet/common_pet_names'
);

$form['pet_weight'] = array(
  '#title' => t('Weight'),
  '#type' => 'textfield',
  '#description' => t('Enter the weight of your pet in kilograms.'),
  '#after_field' => t('kilograms'),
  '#default_value' => $user->pet_weight,
  '#size' => 4,
  '#weight' => 10
);
```

Результат показан на рис. 11.11.

Рис. 11.11. Элемент *textfield*

Свойства *#field_prefix* и *#field_suffix* являются специфичными для текстовых полей и помещают строку сразу же до или после текстового поля ввода.

Свойство *#autocomplete* определяет путь, по которому автоматически включенный JavaScript-код системы Drupal будет отправлять HTTP-запросы, используя jQuery. В предыдущем примере будет запрашиваться `http://example.com/?q=pet/common_pet_names`. За рабочим примером обращайтесь к функции `user_autocomplete()` в `modules/user/user.pages.inc`.

Обычно с элементом текстового поля используются следующие свойства: *#attributes*, *#autocomplete_path* (значение по умолчанию — FALSE), *#default_value*, *#description*, *#field_prefix*, *#field_suffix*, *#maxlength* (значение по умолчанию — 128), *#prefix*, *#required*, *#size* (значение по умолчанию — 60), *#suffix*, *#title*, *#process* (значение по умолчанию — `form_expandahah`) и *#weight*.

Элемент *password*

Этот элемент создает HTML-поле ввода пароля, в котором вводимые пользователем символы не отображаются (обычно вместо них отображаются точки). Ниже показан пример из `user_login_block()`:

```
$form['pass'] = array('#type' => 'password',
  '#title' => t('Password'),
  '#maxlength' => 60,
  '#size' => 15,
  '#required' => TRUE,
);
```

Обычно с элементом ввода пароля используются следующие свойства: `#attributes`, `#description`, `#maxlength`, `#prefix`, `#required`, `#size` (значение по умолчанию — 60), `#suffix`, `#title`, `#process` (значение по умолчанию — `form_expand_ahah`) и `#weight`. Свойство `#default_value` с элементом `password` не используется по причинам, связанным с безопасностью.

Элемент *password_confirm*

Этот элемент создает два HTML-поля ввода пароля и присоединяет функцию проверки на совпадение двух введенных значений. Например, этот элемент используется модулем `user`, когда пользователь изменяет свой пароль.

```
$form['account']['pass'] = array(
  '#type' => 'password_confirm',
  '#description' => t('To change the current user password, enter the new
    password in both fields.'),
  '#size' => 25,
);
```

Элемент *textarea*

Пример элемента `textarea` представлен ниже:

```
$form['pet_habits'] = array(
  '#title' => t('Habits'),
  '#type' => 'textarea',
  '#description' => t('Describe the habits of your pet.'),
  '#default_value' => $user->pet_habits,
  '#cols' => 40,
  '#rows' => 3,
  '#resizable' => FALSE,
  '#weight' => 15
);
```

Обычно с элементом `textarea` используются следующие свойства: `#attributes`, `#cols` (значение по умолчанию — 60), `#default_value`, `#description`, `#prefix`, `#required`, `#resizable`, `#suffix`, `#title`, `#rows` (значение по умолчанию — 5), `#process` (значение по умолчанию — `form_expand_ahah`) и `#weight`.

Значение `#cols` может оказаться недействительным, если включено использование динамического изменения текстовой области, что реализуется установкой `#resizable` в `TRUE`.

Элемент *select*

Ниже представлен пример элемента `select` из `modules/statistics/statistics.admin.inc`:


```

$period = drupal_map_assoc(array(3600, 10800, 21600, 32400, 43200, 86400,
172800, 259200, 604800, 1209600, 2419200, 4838400, 9676800), 'format_interval');
/* Период сейчас выглядит так:
Array (
  [3600] => 1 час
  [10800] => 3 часа
  [21600] => 6 часов
  [32400] => 9 часов
  [43200] => 12 часов
  [86400] => 1 день
  [172800] => 2 дня
  [259200] => 3 дня
  [604800] => 1 неделя
  [1209600] => 2 недели
  [2419200] => 4 недели
  [4838400] => 8 недель
  [9676800] => 16 недель )
*/
$form['access']['statistics_flush_accesslog_timer'] = array(
  '#type' => 'select',
  '#title' => t('Discard access logs older than'),
  '#default_value' => variable_get('statistics_flush_accesslog_timer',
259200),
  '#options' => $period,
  '#description' => t('Older access log entries (including referrer
statistics) will be automatically discarded. (Requires
a correctly configured


```

Drupal поддерживает группирование в опциях выбора, определяя свойство `#options` как ассоциативный массив элементов подменю (рис. 11.12).

```

$options = array(
  array(
    t('Healthy') => array(
      1 => t('wagging'),
      2 => t('upright'),
      3 => t('no tail')
    ),
  ),
  array(
    t('Unhealthy') => array(
      4 => t('bleeding'),
      5 => t('oozing')
    ),
  ),
);
$form['pet_tail'] = array(
  '#title' => t('Tail demeanor'),
  '#type' => 'select',
  '#description' => t('Pick the closest match that describes the tail of
your pet.'),
  '#options' => $options,
  '#multiple' => FALSE,
  '#weight' => 20
);

```



Рис. 11.12. Элемент select, использующий группирование выбора

Возможность множественного выбора включается установкой свойства `#multiple` в `TRUE`. Это также изменяет значение `$form_state['values']` со строки (например, `'pet_tail' = '2'`, предполагая выбор `upright` в предыдущем примере) на массив значений (например, `pet_tail = array( 1 => '1', 2 => '2')`, если в предыдущем примере были выбраны сразу `wagging` и `upright`).

Обычно с элементом `select` используются следующие свойства: `#attributes`, `#default_value`, `#description`, `#multiple`, `#options`, `#prefix`, `#required`, `#suffix`, `#title`, `#process` (значение по умолчанию — `form_expandahah`) и `#weight`.

Элемент *radios*

Ниже приведен пример применения элементов `radios` (переключателей) из `modules/block/block.admin.inc`:

```
$form['user_vis_settings']['custom'] = array(
  '#type' => 'radios',
  '#title' => t('Custom visibility settings'),
  '#options' => array(
    t('Users cannot control whether or not they see this block.'),
    t('Show this block by default, but let individual users hide it.'),
    t('Hide this block by default but let individual users show it.')
  ),
  '#description' => t('Allow individual users to customize the visibility
of this block in their account settings.'),
  '#default_value' => $edit['custom'],
);
```

Обычно с элементом `radios` используются следующие свойства: `#attributes`, `#default_value`, `#description`, `#options`, `#prefix`, `#required`, `#suffix`, `#title` и `#weight`. Обратите внимание, что свойство `#process` по умолчанию установлено в `expand_radios()` (см. `includes/form.inc`).

Элемент *checkboxes*

Пример применения элементов `checkboxes` (флажков) представлен ниже. Визуализированная версия этого элемента показана на рис. 11.13.

```
$options = array(
  'poison' => t('Sprays deadly poison'),
  'metal' => t('Can bite/claw through metal'),
  'deadly' => t('Killed previous owner') );
$form['danger'] = array(
  '#title' => t('Special conditions'),
  '#type' => 'checkboxes',
  '#description' => (t('Please note if any of these conditions apply to your pet.')),
  '#options' => $options,
  '#weight' => 25
);
```

Special conditions:

- ☒ Sprays deadly poison
- ☒ Can bite/claw through metal
- ☐ Killed previous owner

Please note if any of these conditions apply to your pet.

Рис. 11.13. Пример использования флажков

Функция `array_filter()` часто применяется в функциях проверки достоверности и отправки для получения ключей отмеченных флажков `checkboxes`. Например, если первые два флажка на рис. 11.13 отмечены, то `$form_state['values']['danger']` будет содержать следующее:

```
array(
  'poison' => 'poison',
  'metal' => 'metal',
  'deadly' => 0,
)
```

Выполнение `array_filter($form_state['values']['danger'])` дает в результате массив, содержащий ключи отмеченных флажков: `array('poison', 'metal')`.

Обычно с элементом `checkboxes` используются следующие свойства: `#attributes`, `#default_value`, `#description`, `#options`, `#prefix`, `#required`, `#suffix`, `#title`, `#tree` (значение по умолчанию — `TRUE`) и `#weight`. Обратите внимание, что свойство `#process` по умолчанию установлено в `expand_checkboxes()` (см. `includes/form.inc`).

Элемент *value*

Элемент `value` используется, чтобы внутренне передать значения из `$form` в `$form_state['values']` без отправки браузеру, например:

```
$form['pid'] = array(
  '#type' => 'value',
  '#value' => 123
);
```

При отправке формы `$form_state['values']['pid']` будет равно 123.

Не путайте выражения `#type = 'value'` и `#value = 123`. Первое объявляет тип элемента, а второе — его значение. С элементом `value` могут использоваться лишь свойства `#type` и `#value`.

Элемент *hidden*

Этот элемент используется для передачи скрытого значения в форму с помощью поля ввода HTML типа `hidden`, как показано в следующем примере:

```
$form['my_hidden_field'] = array(
  '#type' => 'hidden',
  '#value' => t('I am a hidden field value'),
);
```

Если необходимо отправить скрытое значение через форму, то обычно лучшим вариантом является использование для этого элемента `value`; применение элемента `hidden` оправдано только в случаях, когда `value` оказывается недостаточно. Причина в том, что пользователь может увидеть скрытый элемент в HTML-коде веб-формы, а элемент `value` является внутренним для Drupal и в HTML-код не включается.

С элементом `hidden` используются только свойства `#prefix`, `#suffix`, `#process` (значение по умолчанию — `form_expand_ahah`) и `#value`.

Элемент *date*

Элемент `date` (дата), как показано на рис. 11.14, является составным элементом с тремя полями выбора:

```
$form['deadline'] = array(
  '#title' => t('Deadline'),
  '#type' => 'date',
  '#description' => t('Set the deadline.'),
);
```

```
'#default_value' => array(
  'month' => format_date(time(), 'custom', 'n'),
  'day' => format_date(time(), 'custom', 'j'),
  'year' => format_date(time(), 'custom', 'Y')
)
);
```

Рис. 11.14. Элемент установки даты

Обычно с элементом `date` используются следующие свойства: `#attributes`, `#default_value`, `#description`, `#prefix`, `#required`, `#suffix`, `#title` и `#weight`. Свойство `#process` по умолчанию обеспечивает вызов функции `expand_date()`, в которой поле выбора года жестко закодировано для диапазона с 1900 по 2050 гг. Свойство `#validate` по умолчанию установлено в `date_validate()` (обе функции могут быть найдены в `includes/form.inc`). Определить эти свойства можно при определении элемента даты в форме, чтобы использовать собственный код.

Элемент *weight*

Элемент `weight` (не путать со свойством `#weight`) — это раскрывающийся список, используемый для указания веса:

```
$form['weight'] = array('#type' => 'weight',
  '#title' => t('Weight'),
  '#default_value' => $edit['weight'],
  '#delta' => 10,
  '#description' => t('In listings, the heavier vocabularies will sink and
    the lighter vocabularies will be positioned nearer the top.'),
);
```

Приведенный выше код будет визуализирован так, как показано на рис. 11.15.

Рис. 11.15. Элемент *weight*

Свойство `#delta` определяет диапазон значений веса, которые можно выбрать; значение по умолчанию равно 10. Например, если установить `#delta` в 50, диапазон значений веса будет составлять от -50 до 50. Обычно с элементом `weight` используются следующие свойства: `#attributes`, `#delta` (значение по умолчанию — 10), `#default_value`, `#description`, `#prefix`, `#required`, `#suffix`, `#title` и `#weight`. Свойство `#process` по умолчанию установлено в `array('process_weight', 'form_expand_ahah')`.

Загрузка файлов

Элемент `file` создает интерфейс загрузки файлов. Ниже приведен пример из `modules/user/user.module`:

```
$form['picture']['picture_upload'] = array(
  '#type' => 'file',
  '#title' => t('Upload picture'),
  '#size' => 48,
  '#description' => t('Your virtual face or picture.')
);
```

Визуализация этого элемента показана на рис. 11.16.

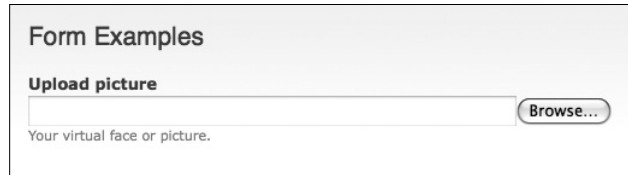


Рис. 11.16. Элемент загрузки файлов

Обратите внимание, что в случае использования элемента `file` необходимо установить свойство `enctype` в корневом элементе формы:

```
$form['#attributes']['enctype'] = 'multipart/form-data';
```

Обычно с элементом `file` используются следующие свойства: `#attributes`, `#default_value`, `#description`, `#prefix`, `#required`, `#size` (значение по умолчанию — 60), `#suffix`, `#title` и `#weight`.

Элемент *fieldset*

Элемент `fieldset` используется для группирования элементов. Он может быть объявлен сворачиваемым (свойство `#collapsible` установлено в `TRUE`); это означает, что JavaScript-код, автоматически предоставляемый Drupal, используется для открытия и закрытия набора полей динамически по щелчку кнопкой мыши во время просмотра формы. Обратите внимание на применение свойства `#access` для разрешения или запрета доступа ко всем полям внутри набора.

```
// Нода информации об авторе для администраторов.
$form['author'] = array(
  '#type' => 'fieldset',
  '#access' => user_access('administer nodes'),
  '#title' => t('Authoring information'),
  '#collapsible' => TRUE,
  '#collapsed' => TRUE,
  '#weight' => 20,
);
```

Обычно с элементом `fieldset` используются следующие свойства: `#attributes`, `#collapsed` (значение по умолчанию — `FALSE`), `#collapsible` (значение по умолчанию — `FALSE`), `#description`, `#prefix`, `#suffix`, `#title`, `#process` (значение по умолчанию — `form_expandahah`) и `#weight`.

Элемент *submit*

Элемент `submit` используется для отправки формы. Слово, отображаемое по умолчанию на кнопке — это `Submit` (Отправить), но его легко изменить с помощью свойства `#value`:

```
$form['submit'] = array(
  '#type' => 'submit',
  '#value' => t('Continue'),
);
```

Обычно с элементом `submit` используются следующие свойства: `#attributes`, `#button_type` (значение по умолчанию — `'submit'`), `#executes_submit_callback` (значение по умолчанию — `TRUE`), `#name` (значение по умолчанию — `'op'`), `#prefix`, `#suffix`, `#value`, `#process` (значение по умолчанию — `form_expandahah`) и `#weight`.

Добавок свойства `#validate` и `#submit` могут быть установлены непосредственно в элемент `submit`. Например, если `#submit` установлено в `array('my_special_form_submit')`, то функция `my_special_form_submit()` будет применяться вместо определенных формой обработчиков отправки.

Элемент *button*

Элемент `button` (кнопка) — такой же самый, как элемент `submit`, за исключением того, что значение свойства `#executes_submit_callback` по умолчанию равно `FALSE`. Это свойство сообщает системе Drupal, должна ли она обработать форму (`TRUE`) или просто повторно визуализировать ее (`FALSE`). Как и в случае `submit`, элементу `button` можно назначать специфические функции проверки достоверности и отправки.

Элемент *image_button*

Элемент `image_button` (кнопка с изображением) — это то же самое, что элемент `submit`, но с двумя исключениями. Во-первых, он имеет свойство `#src`, содержащее URL изображения. Во-вторых, он устанавливает внутреннее свойство формы `#has_garbage_value` в `TRUE`, что предотвращает использование `#default_value` из-за ошибки, присутствующей в браузере Microsoft Internet Explorer. Не применяйте `#default_value` с кнопками с изображениями. Ниже показан пример кнопки с изображением, использующей встроенную картинку “Powered by Drupal”:

```
$form['my_image_button'] = array(
  '#type' => 'image_button',
  '#src' => 'misc/powered-blue-80x15.png',
  '#value' => 'foo',
);
```

Значение кнопки может быть безопасно извлечено через `$form_state['clicked_button']['#value']`.

Элемент *markup*

Элемент `markup` — это тип элемента по умолчанию, если явно не установлено свойство `#type`. Он используется для ввода текста или HTML-кода в середине формы.

```
$form['disclaimer'] = array(
  '#prefix' => '<div>',
  '#value' => t('The information below is entirely optional.'),
  '#suffix' => '</div>'
);
```

Обычно с элементом `markup` используются следующие свойства: `#attributes`, `#prefix` (значение по умолчанию — пустая строка), `#suffix` (значение по умолчанию — пустая строка), `#value` и `#weight`.

Внимание! Если вы вводите текст в сворачиваемом наборе полей, поместите его в дескриптор `<div>`, как показано в примере, чтобы, когда набор полей сворачивается, текст также сворачивался вместе с ним.

Элемент *item*

Элемент *item* имеет такой же формат, как у элементов ввода других типов, таких как *textfield* или *select*, однако в нем отсутствует поле ввода:

```
$form['removed'] = array(
  '#title' => t('Shoe size'),
  '#type' => 'item',
  '#description' => t('This question has been removed because the law
    prohibits us from asking your shoe size.')
);
```

Этот элемент визуализируется так, как показано на рис. 11.17.

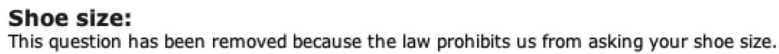


Рис. 11.17. Элемент *item*

Обычно с элементом *item* используются следующие свойства: *#attributes*, *#description*, *#prefix* (значение по умолчанию — пустая строка), *#required*, *#suffix* (значение по умолчанию — пустая строка), *#title*, *#value* и *#weight*.

Свойство *#ahah*

Свойство элемента *#ahah* предоставляет информацию реализации Drupal асинхронного HTML и HTTP (Asynchronous HTML and HTTP — АНАН), которая позволяет изменять элементы формы посредством кода JavaScript.

Совет. Возможно, вы заметили, что у многих элементов форм, которые мы описали, значение по умолчанию для *#process* равно *form_expand_ahah*. Присутствие свойства *#ahah* в элементе указывает Drupal, что с этим элементом будет использоваться АНАН. Функция *form_expand_ahah()* гарантирует разумные значения по умолчанию для *#ahah*.

Ниже приведен пример использования кнопки *Attach* (Присоединить), предоставленной модулем *upload* для загрузки файлов:

```
$form['new']['attach'] = array(
  '#type' => 'submit',
  '#value' => t('Attach'),
  '#name' => 'attach',
  '#ahah' => array(
    'path' => 'upload/js',
    'wrapper' => 'attach-wrapper',
    'progress' => array(
      'type' => 'bar',
      'message' => t('Please wait...'),
    ),
  ),
  '#submit' => array('node_form_submit_build_node'),
);
```

Значением свойства *#ahah* является массив с ключами. Ниже перечислены обязательные ключи.

- *path*. Путь Drupal к элементу меню, который будет запрашивать JavaScript-код. Обратный вызов элемента меню и путь к элементу меню оканчиваются на *js*;

это указывает, что элемент вызывается из JavaScript-кода. В предыдущем примере путь Drupal выглядит как `upload/js`, а соответствующий обратный вызов — как `upload_js()` (вы можете убедиться в этом, заглянув в код функции `upload_menu()` внутри `modules/upload/upload/module`).

- `wrapper`. Соответствует атрибуту `id` элемента HTML (обычно `<div>`). В предыдущем примере модуль `upload` ссылается на следующий элемент: `<div id="attach-wrapper">`.

Следующие ключи являются необязательными.

- `effect`. Визуальный эффект для использования при замене элемента. Возможные значения: `none`, `fade` и `slide`. По умолчанию принимается `none`.
- `event`. Событие, которое инициирует выполнение браузером HTTP-запроса JavaScript. Drupal устанавливает значения по умолчанию на основе типа элемента. Эти значения перечислены в табл. 11.1.

Таблица 11.1. Имена по умолчанию событий, которые инициируют АНАН в элементах формы

Элемент	Событие по умолчанию
<code>submit</code>	<code>mousedown</code> *
<code>button</code>	<code>mousedown</code> *
<code>image_button</code>	<code>mousedown</code> *
<code>password</code>	<code>blur</code>
<code>textfield</code>	<code>blur</code>
<code>textarea</code>	<code>blur</code>
<code>radio</code>	<code>change</code>
<code>checkbox</code>	<code>change</code>
<code>select</code>	<code>change</code>

* Также добавляется и событие `keypress`.

- `method`. Метод jQuery, который будет использован для изменения существующего HTML-кода, когда вернется ответ на HTTP-запрос JavaScript. Возможные значения: `after`, `append`, `before`, `prepend` и `replace`. По умолчанию принимается метод `replace`. Это значение будет применяться в следующем JavaScript-коде (см. `misc/ahah.js`):

```
if (this.method == 'replace') {
  wrapper.empty().append(new_content);
}
else {
  wrapper[this.method](new_content);
}
```

- `progress`. Это способ, которым Drupal сигнализирует пользователю, что произошло событие JavaScript. Значение этого свойства — массив с ключами `type` и `message`, например:

```
$form['submit'] = array(
  '#type' => 'submit',
  '#value' => t('Click Me'),
  '#ahah' => array(
    'event' => 'click',
```



```

    'path' => 'poof/message_js',
    'wrapper' => 'target',
    'effect' => 'fade',
    'progress' => array(
        'type' => 'throbber',
        'message' => t('One moment...'),
    ),
)
);

```

Значением по умолчанию для `type` является `throbber` — вращающаяся анимированная пиктограмма, которая отображает необязательное сообщение, пока выполняется HTTP-запрос JavaScript. Другой выбор — `bar`, представляющий собой панель хода работ (в этом случае добавляется отдельный файл JavaScript по имени `misc/progress.js`). Если тип установлен в `bar`, доступны следующие необязательные ключи: `url` и `interval`. Ключ `url` указывает URL для панели хода работ, чтобы обратиться к ней для определения процента выполнения от 0 до 100, а `interval` задает частоту проверки хода работ (в секундах).

- `selector`. Указание `selector` — это способ присоединения результата HTTP-запроса JavaScript к множеству элементов на странице (а не только к элементу формы).

Рассмотрим краткий пример формы, которая позволяет динамическую замену некоторого текста с применением АНАН. Кнопка использует `throbber` для указания, что пользователь должен ожидать (рис. 11.18). Ниже приведен текст `sites/all/modules/custom/poof/poof.info`.

```

name = Poof
description = Demonstrates АНАН forms.
package = Pro Drupal Development
core = 7.x
files[]=poof.module

```

А вот как выглядит содержимое `sites/all/modules/custom/poof/poof.module`:

```

<?php
/**
 * Реализация hook_menu().
 */
function poof_menu() {
    $items['poof'] = array(
        'title' => 'Ahah!',
        'page callback' => 'drupal_get_form',
        'page arguments' => array('poof_form'),
        'access callback' => TRUE,
        'type' => MENU_NORMAL_ITEM,
    );
    $items['poof/message_js'] = array(
        'page callback' => 'poof_message_js',
        'type' => MENU_CALLBACK,
        'access callback' => TRUE,
    );
    return $items;
}

/**
 * Определение формы.
 */

```

```
function poof_form() {
  $form['target'] = array(
    '#type' => 'markup',
    '#prefix' => '<div id="target">',
    '#value' => t('Click the button below. I dare you.'),
    '#suffix' => '</div>',
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Click Me'),
    '#ahah' => array(
      'event' => 'click',
      'path' => 'poof/message_js',
      'wrapper' => 'target',
      'effect' => 'fade',
    )
  );
  return $form;
}

/**
 * Обратный вызов меню для дополнений АНАН.
 */
function poof_message_js() {
  $output = t('POOF!');
  drupal_json(array('status' => TRUE, 'data' => $output));
}
```

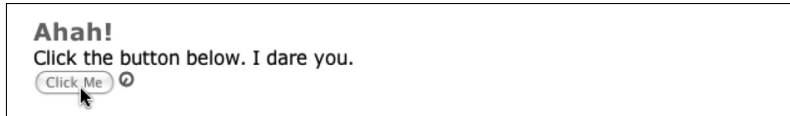


Рис. 11.18. Щелчок на кнопке приведет к основанной на АНАН замене текста после отображения вращающейся анимированной пиктограммы throbber

Ниже приведен тот же модуль, но на этот раз реализованный с использованием панели хода работ, обновляемой каждые две секунды (рис. 11.19).

Внимание! Этот модуль просто демонстрирует то, как взаимодействовать с панелью хода работ; в реальном модуле выдавался бы процент завершения текущей задачи. В частности, нельзя было бы использовать систему хранения переменных Drupal для сохранения и чтения информации о ходе работ, как в этом примере, поскольку возникла бы путаница со значениями для множества пользователей, одновременно запускающих форму. Вместо этого нужно было бы выполнять запрос к базе данных.

```
<?php

/**
 * Реализация hook_menu().
 */
function poof_menu() {
  $items['poof'] = array(
    'title' => 'Ahah!',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('poof_form'),
    'access callback' => TRUE,
```

```

    'type' => MENU_NORMAL_ITEM,
  );
  $items['poof/message_js'] = array(
    'page callback' => 'poof_message_js',
    'type' => MENU_CALLBACK,
    'access callback' => TRUE,
  );
  $items['poof/interval_js'] = array(
    'page callback' => 'poof_interval_js',
    'type' => MENU_CALLBACK,
    'access callback' => TRUE,
  );
  return $items;
}

/**
 * Определение формы.
 */
function poof_form() {
  $form['target'] = array(
    '#type' => 'markup',
    '#prefix' => '<div id="target">',
    '#value' => t('Click the button below. I dare you.'),
    '#suffix' => '</div>',
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Click Me'),
    '#ahah' => array(
      'event' => 'click',
      'path' => 'poof/message_js',
      'wrapper' => 'target',
      'effect' => 'fade',
      'progress' => array(
        'type' => 'bar',
        'message' => t('One moment...'),
        'interval' => 2,
        'url' => 'poof/interval_js',
      ),
    ),
  );
  return $form;
}

/**
 * Обратный вызов меню для дополнений АНАН.
 */
function poof_message_js() {
  $output = t('POOF!');
  for ($i = 0; $i < 100; $i = $i + 20) {
    // Записать, насколько далеко мы продвинулись.
    variable_set('poof_percentage', $i);
    // Эмулировать выполнение задачи двухсекундным ожиданием.
    sleep(2);
  }
  drupal_json(array('status' => TRUE, 'data' => $output));
}

```

```

/**
 * Обратный вызов для интервалов панели хода работ АНАН.
 */
function roof_interval_js() {
  // Прочитать, насколько далеко мы продвинулись.
  $percentage = variable_get('roof_percentage', 0);

  // Вернуть значение для панели хода работ JavaScript.
  drupal_json(array('percentage' => $percentage));
}

```

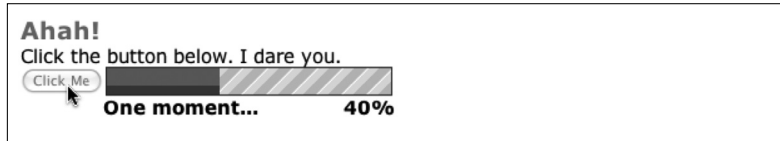


Рис. 11.19. Панель хода работ показывает процент выполнения

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Понимать, как работает Form API.
- Уметь создавать простые формы.
- Понимать, как изменять визуализируемую форму с использованием функции темы.
- Знать, как пишется функция проверки достоверности для формы или для ее отдельных элементов.
- Знать, как пишется функция отправки и как выполняется перенаправление после обработки формы.
- Понимать, как изменять существующие формы.
- Уметь создавать динамические формы.
- Знать, какие свойства определения формы можно использовать, и что они означают.
- Знать, какие элементы формы (текстовые поля, поля выбора, переключатели и т.д.) доступны в Drupal.
- Понимать, как работает в формах замена текста, основанная на АНАН.

Для получения дополнительной информации о формах, включая полезные советы и трюки, обращайтесь к руководству по Drupal, которое находится по адресу <http://drupal.org/node/37775>.

ГЛАВА 12

Манипулирование данными, вводимыми пользователем: система фильтров

Добавление контента на веб-сайт может превратиться в рутинную работу, когда приходится форматировать информацию вручную. С другой стороны, для того чтобы введенный текст хорошо смотрелся на веб-сайте, необходимо знание HTML, овладеть которым большинство пользователей совершенно не торопится. Даже тем, кто знает HTML, зачастую трудно на ходу вставлять дескрипторы в ситуациях, требующих быстрого реагирования. Дескрипторы абзаца, дескрипторы ссылки, дескрипторы разрыва строки — в общем, их довольно много. К счастью, в Drupal используются предварительно подготовленные процедуры, называемые *фильтрами*, которые превращают ввод данных в простой и эффективный процесс. Фильтры выполняют такие манипуляции с текстом, как создание URL, реагирующих на щелчки, преобразование разрывов строк в дескрипторы `<p>` и `<br />` и даже удаление вредоносного HTML-кода. Создание фильтров и манипуляции с отправленными пользователем данными основано на механизме `hook_filter()`.

Фильтры

Фильтры — это почти всегда единственное действие, такое как “удалить все гиперссылки”, “добавить случайное изображение к сообщению” или даже “перевести это на пиратский сленг” (см. `pirate.module` на странице <http://drupal.org/project/pirate>). Как показано на рис. 12.1, они принимают некоторый текстовый ввод, проводят над ним определенные манипуляции и возвращают вывод.

Обычное применение фильтров состоит в удалении нежелательной разметки из пользовательского ввода. На рис. 12.2 показан HTML-фильтр Drupal в действии.



Рис. 12.1. Фильтр трансформирует текст некоторым образом и возвращает преобразованный текст

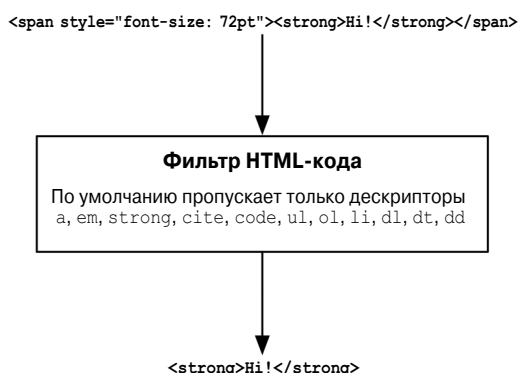


Рис. 12.2. Фильтр Limit allowed HTML tags (Ограничить разрешенные HTML-дескрипторы) пропускает только определенные дескрипторы. Этот фильтр важен для предотвращения межсайтовых атак посредством сценариев

Фильтры и текстовые форматы

Попытку найти список установленных фильтров в пределах интерфейса администратора нельзя отнести к числу интуитивно понятных: предполагается, что вы должны хорошо понимать, что делает тот или иной фильтр, чтобы знать, где его искать. Для того чтобы фильтр мог выполнять свою работу, ему необходимо назначить *текстовый формат* Drupal, как показано на рис. 12.3. Текстовые форматы группируют фильтры, так что в процессе обработки контента они могут выполняться как один пакет. Это намного проще, чем выбор подходящих фильтров для каждой отправки. Для просмотра списка установленных фильтров необходимо либо перейти к конфигурированию существующего текстового формата, либо создать новый формат, щелкнув на ссылке Configuration (Конфигурация) в верхней части страницы, затем на ссылке Text formats (Текстовые форматы) на странице Configuration и, наконец, на ссылке Add text format (Добавить текстовый формат).

Совет. Текстовый формат Drupal состоит из коллекции фильтров.

Среда Drupal поставляется с четырьмя текстовыми форматами (рис. 12.4).

- Текстовый формат Filtered HTML (Фильтрованный HTML), состоящий из четырех фильтров:
 - фильтр Limit allowed HTML tags (Ограничить разрешенные HTML-дескрипторы), который ограничивает перечень HTML-дескрипторов, пропускаемых фильтром;
 - фильтр Convert URLs into links (Преобразовать URL в ссылки), который преобразует веб-адреса и адреса электронной почты в гиперссылки;
 - фильтр Convert line breaks into HTML line break (Преобразовать разрывы строк в дескрипторы разрывов строк HTML), который превращает возвраты каретки в их HTML-аналоги;
 - фильтр Correct faulty and chopped off HTML (Корректировать ошибочную и разорванную HTML-разметку).

- Текстовый формат Full HTML (Полный HTML) не использует фильтр Limit allowed HTML tags, но задействует фильтры Convert URLs into links, Convert line breaks into HTML line break и Correct faulty and chopped off HTML.
- Текстовый формат Plain Text (Простой текст) отображает HTML-дескрипторы как простой текст.
- Текстовый формат PHP Code (PHP-код) образован из фильтра, обращающегося к анализатору PHP, и его задача — выполнить любой PHP-код, который содержится в сообщении. Хорошее эмпирическое правило гласит: никогда не предоставлять пользователям текстовый формат, который использует анализатор PHP. Если разрешить пользователям выполнять PHP-код, они смогут сделать все, на что способен PHP, включая уничтожение сайта и удаление всех существующих данных. Именно потому Drupal поставляется с отключенным фильтром анализатора PHP. Если его включать все-таки приходится, необходимо активировать модуль фильтра PHP.

Внимание! Разрешение текстового формата PHP Code для любого пользователя, посещающего сайт, является дополнительным риском в плане безопасности. Лучше как можно меньше использовать этот текстовый формат, а если уж деваться некуда, то свести его применение к минимуму, причем только для суперпользователя (пользователя с идентификатором 1).

Add text format ⓘ

Dashboard » Configuration » Content authoring » Text formats

Name *

Roles

☐ anonymous user

☐ authenticated user

☒ administrator

Enabled filters

☐ Limit allowed HTML tags

☐ Display any HTML as plain text

☐ Convert URLs into links

☐ Convert line breaks into HTML (i.e. `<br>` and `<p>`)

☐ PHP evaluator
Executes a piece of PHP code. The usage of this filter should be restricted to administrators only!

☐ Correct faulty and chopped off HTML

Filter processing order

Filter settings

Save configuration

Рис. 12.3. Установленные фильтры перечислены на форме Add text format (Добавить текстовый формат)

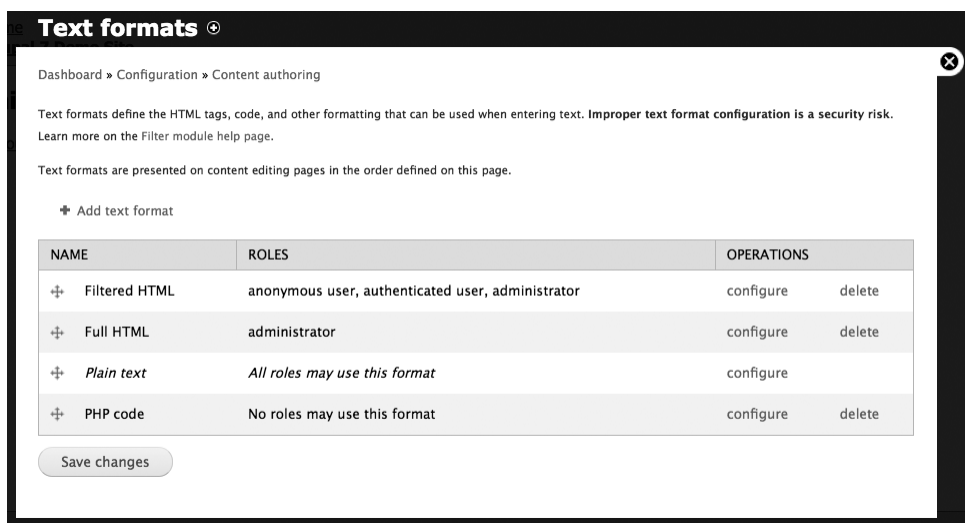


Рис. 12.4. По умолчанию Drupal устанавливается с четырьмя конфигурируемыми текстовыми форматами

Поскольку текстовые форматы — это коллекции фильтров, они расширяемы. Фильтры можно добавлять и удалять, как показано на рис. 12.5. Можно также изменять название текстового формата, добавить фильтр, удалить фильтр или даже изменить порядок, в котором выполняются фильтры текстового формата, чтобы избежать конфликтов. Например, может потребоваться выполнить URL-фильтр до фильтра *Correct faulty and chopped off HTML*, чтобы последний просмотрел дескрипторы `<a>`, созданные URL-фильтром.

На заметку! Текстовые форматы (группы фильтров) управляются на уровне интерфейса. При определении нового фильтра разработчики не должны волноваться о текстовых форматах. Эта работа возлагается на администратора сайта Drupal.

Инсталляция фильтра

Инсталляция фильтра следует той же самой процедуре, что и инсталляция модуля, поскольку фильтры находятся в файлах модуля. Поэтому манипулирование доступностью фильтра сводится к включению или отключению соответствующего модуля щелчком на ссылке *Modules* (Модули) в верхнем меню. После инсталляции щелкните на ссылке *Configuration* (Конфигурация) в верхней части страницы и на странице *Configuration* щелкните на ссылке *Text formats* (Текстовые форматы), чтобы назначить новый фильтр выбранному текстовому формату (или форматам). На рис. 12.6 показано отношение между фильтрами и модулями.

Когда нужно использовать фильтры

Вы могли бы задаться вопросом, зачем вообще нужна система фильтров, если можно легко управлять текстом, используя повсеместно существующие хуки. Например, вместо фильтра URL, казалось бы, удобно использовать хук `hook_node_view()` для преобразования URL в реагирующие на щелчки ссылки.

Текстовый формат**Фильтры текстового формата**

Filtered HTML (Фильтрованный HTML)	URL Filter (Фильтр URL)
	HTML Filter (Фильтр HTML)
	Line Break Filter (Фильтр разрывов строк)
	HTML Corrector (Корректор HTML)
Full HTML (Полный HTML)	URL Filter (Фильтр URL)
	Line Break Filter (Фильтр разрывов строк)
	HTML Corrector (Корректор HTML)
Plain Text (Простой текст)	HTML As Plain Text Filter (Фильтр преобразования HTML в простой текст)
	URL Filter (Фильтр URL)
	Line Break Filter (Фильтр разрывов строк)
PHP Code (PHP-код)	PHP Evaluator (Анализатор PHP)

Рис. 12.5. Текстовые форматы представляют собой коллекции фильтров. Здесь показаны четыре текстовых формата Drupal по умолчанию

Текстовый формат**Фильтры****Инсталлируются модулем**

Специальный формат	URL Filter (Фильтр URL)	filter.module
	Pirate Filter (Фильтр пиратского сленга)	pirate.module
	HTML Filter (Фильтр HTML)	filter.module
	Line Break Filter (Фильтр разрывов строк)	filter.module
	HTML Corrector (Корректор HTML)	filter.module

Рис. 12.6. Фильтры создаются как части модулей

Рассмотрим случай, когда имеется пять различных фильтров, которые нужно выполнить для поля тела ноды. Далее предположим, что просматривается страница `http://example.com/?q=node`, на которой одновременно отображается десять нод. Это означает, что для генерации одностраничного представления потребуется выполнить 50 фильтров, и фильтрация текста может превратиться в весьма дорогостоящую операцию. Это также означает, что при каждом обращении нода должна проходить через эти фильтры, даже если фильтруемый текст не менялся. Эта операция выполнялась бы много раз, хотя без нее можно обойтись.

Система фильтров имеет уровень кэширования, который обеспечивает существенное улучшение производительности. Если для данного фрагмента текста были выполнены все фильтры, отфильтрованная версия этого текста сохраняется в таблице `cache_filter`, и она остается в кэше до тех пор, пока текст не будет опять изменен (модификация обнаруживается за счет хеширования отфильтрованного контента по алгоритму `sha256`). Возвращаясь к нашему примеру, отметим, что загрузка десяти нод могла бы эффективно обойти все фильтры и загрузить данные прямо из таблицы кэша, если текст не изменялся, причем намного быстрее.

Совет. `sha256` — это алгоритм вычисления хеш-значения для строки текста. Система Drupal использует его как эффективный индексный столбец в базе данных для нахождения отфильтрованных данных ноды.

Теперь можно было бы сказать: “А что если повторно сохранить отфильтрованный текст в таблице нод в нашем хуке `hook_node_view()`? Тогда он вел бы себя точно так же, как система фильтров”. Хотя это и относится к вопросам производительности, была бы нарушена фундаментальная концепция архитектуры Drupal: *никогда не изменять исходные данные пользователя*. Представьте, что один из ваших неопытных пользователей возвращается, чтобы отредактировать сообщение, и обнаруживает его в угловых скобках HTML. Скорее всего, он обратится в службу технической поддержки. Цель системы фильтров состоит в том, чтобы оставить исходные данные нетронутыми и в то же время сделать кэшируемые копии отфильтрованных данных доступными остальной части Drupal. Вы увидите, что этот принцип соблюдается и другими API-интерфейсами Drupal.

На заметку! Система фильтров кэширует свои данные, даже если кэширование в Drupal не разрешено на уровне страницы. Если вы видите устаревшие отфильтрованные данные, попробуйте очистить таблицу `cache_filter`, щелкнув на кнопке **Clear cached data** (Очистить кэшированные данные) в верхней части страницы **Configuration** ⇒ **Performance** (Конфигурация ⇒ Производительность).

Создание специального фильтра

Разумеется, фильтры Drupal могут создавать ссылки, форматировать контент и преобразовывать текст в пиратский сленг на лету, но было бы совсем здорово, если бы они могли делать записи в блогах или, по крайней мере, помогать нам в подобном творчестве. Естественно, Drupal может делать это. Давайте создадим модуль с фильтром для вставки случайных фраз в запись блога. Мы настроим его так, чтобы, когда ваши мысли иссякнут, и понадобится творческий всплеск, можно было бы просто напечатать `[juice!]`, которое при сохранении записи будет заменено случайным высказыванием. Мы также сделаем так, чтобы можно было использовать дескриптор `[juice!]` несколько раз в одном сообщении.

Создайте папку по имени `creativejuice` в каталоге `sites/all/modules/custom/`. Сначала добавьте в папку `creativejuice` файл `creativejuice.info`:

```
name = Creative Juice
description = "Adds a random sentence filter to content."
package = Pro Drupal Development
core = 7.x
files[] = creativejuice.module
php = 5.2
```

Затем создайте файл `creativejuice.module` и тоже добавьте его в эту папку:

```
<?php

/**
 * @file
 * Маленький модуль для помощи ловким писателям-романистам, которые
 * получают вдохновение с помощью генератора случайных фраз.
 */
```

Реализация `hook_filter_info()`

Теперь, когда составляющие модуля размещены по своим местам, давайте реализуем `hook_filter_info()` в `creativejuice.module`:

```
/**
 * Реализация hook_filter_info().
 */
function creativejuice_filter_info() {
  $filters = array();
  $filters['creativejuice'] = array(
    'title' => t('Creative Juice filter'),
    'description' => t('Enables users to insert random sentences into their post'),
    'process callback' => '_creativejuice_filter_process',
    'tips callback' => '_creativejuice_filter_tips',
  );
  return $filters;
}
```

Функция обработки

Функция обработки `creativejuice_filter_process` вызывается при каждом сохранении ноды — когда входной тип, установленный для ноды, соответствует текстовому фильтру при условии, что фильтр `creativejuice` включен.

```
/**
 * Обратный вызов для обработки фильтра creativejuice.
 *
 * Здесь выполняется действительная фильтрация. Переданный текст должен
 * быть возвращен после того, как произведены необходимые подстановки.
 */
function _creativejuice_filter_process($text, $filter, $format) {
  while (strpos($text, '[juice!>') !== FALSE) {
    $sentence = creativejuice_sentence();
    $text = preg_replace('&\[juice!\>', $sentence, $text, 1);
  }
  return $text;
}
```

Эта функция относительно проста. Первый шаг состоит в вызове вспомогательной функции, которая возвращает случайную фразу, а во второй строке кода с помощью PHP-функции замены строк производится замена всех вхождений дескриптора [juice!] случайной фразой, возвращенной из `creativejuice_sentence`.

Вспомогательная функция

Вспомогательная функция возвращает случайную фразу, которая будет использоваться фильтром для замены дескриптора [juice!]:

```
/**
 * Генерация случайной фразы.
 */
function creativejuice_sentence() {
    $beginnings = array();
    $beginnings[] = t('A majority of us believe');
    $beginnings[] = t('Generally speaking,');
    $beginnings[] = t('As times carry on');
    $beginnings[] = t('Barren in intellect,');
    $beginnings[] = t('Deficient in insight,');
    $beginnings[] = t('As blazing blue sky pours down torrents of light,');
    $beginnings[] = t('Aloof from the motley throng,');
    $beginnings[] = t('While crafting a new Drupal module,');

    $middles = array();
    $middles[] = t('life flowed in its accustomed stream');
    $middles[] = t('he ransacked the vocabulary');
    $middles[] = t('the grimaces and caperings of buffoonery sting');
    $middles[] = t('the mind freezes at the thought');
    $middles[] = t('reverting to another matter enables freedom');
    $middles[] = t('he lived as modestly as a hermit');
    $middles[] = t('the coder repeatedly invoked hooks');

    $ends = array();
    $ends[] = t('through the red tape of officialdom.');
```

```
$ends[] = t('as it set anew in some fresh and appealing form.');
```

```
$ends[] = t('supported by evidence.');
```

```
$ends[] = t('as fatal as the fang of the most venomous snake.');
```

```
$ends[] = t('as full of spirit as a gray squirrel.');
```

```
$ends[] = t('as dumb as a fish.');
```

```
$ends[] = t('like a damp-handed auctioneer.');
```

```
$ends[] = t('like a bald ferret.');
```

```
$ends[] = t('with a frozen, sharpened badger.');
```

```
$ends[] = t('and achieve CMS nirvana.');
```

```
// Выбрать случайное значение для каждой группы фразы.
$sentence = array(
    $beginnings[mt_rand(0, count($beginnings) - 1)],
    $middles[mt_rand(0, count($middles) - 1)],
    $ends[mt_rand(0, count($ends) - 1)],
);

// Получить три случайных значения из групп фразы,
// соединить их вместе и вернуть фразу.
return implode(' ', $sentence);
}
```

Функция очень проста — она создает массив фраз и случайным образом выбирает фразу для возврата в качестве результата ее вызова.

Хук `creativejuice_filter_tips()` используется для отображения текста справки конечному пользователю. По умолчанию показываются короткие сообщения, ссылающиеся на страницу `http://example.com/?q=filter/tips`, где для каждого фильтра даются более подробные инструкции.

```
/**
 * Обратный вызов для вывода текста справки по фильтру creativejuice.
 *
 * Обратный вызов для вывода текста справки позволяет фильтрам
 * предоставлять пользователям поясняющий текст во время процесса
 * редактирования контента. Короткая справка выводится
 * на экране редактирования контента, в то время как длинная справка
 * предлагается на отдельной странице по ссылке. Применение короткой
 * справки не обязательно, но длинной — настоятельно рекомендуется.
 */
function creativejuice_filter_tips($filter, $format, $long = FALSE) {
  return t('<em>[creativejuice]</em> is replaced with the random sentences.');
```

В предыдущем коде для короткой и для длинной страницы справочной системы возвращается один и тот же текст, но если нужно возвращать более длинный текст пояснения, потребуется проверить параметр `$long`, как показано ниже:

```
function creativejuice_filter_tips($filter, $format, $long = FALSE) {
  if ($long) {
    // Детальное объяснение для http://example.com/?q=filter/tips.
    return t('The Creative Juice filter is for those times when your
      brain is incapable of being creative. These times comes for everyone,
      when even strong coffee and a barrel of jelly beans do not
      create the desired effect. When that happens, you can simply enter
      the [juice!] tag into your posts...');
  }
  else {
    // Краткое пояснение, располагающееся ниже текстового поля сообщения.
    return t('Insert a random sentence into your post with the [juice!] tag.');
```

Как только этот модуль будет активирован на странице модулей, фильтр `creativejuice` становится доступным либо для существующего, либо для нового текстового формата. Например, на рис. 12.7 показано, как выглядит раздел **Text format** (Текстовый формат) формы редактирования ноды после добавления фильтра `creativejuice` к текстовому формату Full HTML.

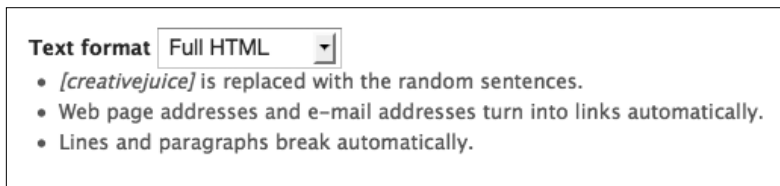


Рис. 12.7. Текстовый формат Full HTML теперь содержит фильтр `creativejuice`

Вы можете создать новую запись блога с правильным текстовым форматом и отправить текст, в котором присутствуют дескрипторы [juice!]:

```
Today was a crazy day. [juice!] Even if that sounds a little odd,  
it still doesn't beat what I heard on the radio. [juice!]
```

После отправки текст приобретает примерно такой вид:

```
Today was a crazy day! Generally speaking, life flowed in its accustomed  
stream through the red tape of officialdom. Even if that sounds a little  
odd, it still doesn't beat what I heard on the radio. Barren in intellect,  
she reverted to another matter like a damp-handed auctioneer.
```

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Понимать, что такое фильтр и текстовый формат и как они используются для преобразования текста.
- Понимать, почему система фильтров более эффективна, чем манипуляции с текстом в других хуках.
- Понимать, как ведут себя текстовые форматы и фильтры.
- Создавать специальные фильтры.
- Понимать, как выполняются различные операции фильтрации.

ГЛАВА 13

Поиск и индексация контента

Модуль поиска Drupal предоставляет развитое решение, которое удовлетворяет потребностям большинства веб-сайтов. В случае если модуль поиска ядра не поддерживает необходимого средства или функциональности, этот модуль можно расширить через Search API. В главе будет показано, как модули могут взаимодействовать с этим API поиска и каким образом создавать специальные формы поиска. Мы также рассмотрим, как Drupal анализирует и индексирует контент, и как можно внедриться в работу индексатора.

Совет. Drupal принимает сложные поисковые запросы, содержащие булевские операции “И”/“ИЛИ”, точные фразы и даже слова, которые нужно исключить, например: Beatles OR “John Lennon” -insect. Здесь производится поиск всех вхождений слова “Beatles” или фразы “John Lennon”, причем результаты не должны включать слово “insect”.

Создание специальной страницы поиска

Drupal обладает встроенной возможностью поиска нод и имен пользователей. Даже когда вы разрабатываете собственные специальные типы нод, система поиска Drupal индексирует контент, который визуализируется в представлении ноды. Предположим, например, что имеется тип ноды recipe (рецепт) с полями ingredients (ингредиенты) и instructions (инструкции) и создается новая нода recipe с идентификатором 22. Поскольку эти поля доступны для просмотра администратором при посещении <http://example.com/?q=node/22>, модуль поиска проиндексирует ноду recipe и ее дополнительные метаданные во время очередного запуска cron.

Хотя на первый взгляд может показаться, что в основе поиска нод и пользователей лежит один и тот же механизм, фактически это — два разных способа расширения поисковой функциональности. Вместо того чтобы делать запрос к таблице node непосредственно для каждого поиска, механизм поиска нод прибегает к помощи индексатора, чтобы обработать контент быстрее и в структурированном формате. Когда поиск нод завершен, делается запрос к структурированным данным, что приводит к заметному ускорению и более точным результатам. Подробнее мы поговорим об индексаторе позже в этой главе.

Поиск имен пользователей не относится к числу сложных, потому что имена пользователей хранятся в единственном столбце базы данных, который и проверяется при поисковом запросе. Кроме того, имена пользователей не могут содержать HTML-код, по-

этому нет необходимости использовать индексатор HTML. Вместо этого можно делать запросы непосредственно к таблице `users` с помощью лишь нескольких строк кода.

В обоих предыдущих случаях модуль поиска Drupal делегирует поиск соответствующему модулю. Простой поиск имени пользователя можно найти в функции `user_search_execute()` из модуля `modules/user/user.module`, в то время как более сложный поиск в нодах осуществляется функцией `node_search_execute()` в `modules/node/node.module`. Здесь важно отметить, что модуль поиска координирует поиск, но делегирует реализации модулям, которым лучше известен контент, где нужно искать.

Форма поиска по умолчанию

В Search API имеется форма поиска по умолчанию, готовая к использованию (рис. 13.1). Если ее интерфейс соответствует вашим потребностям, то все, что потребуется сделать — это реализовать логику, которая находит попадания (hits) для затребованного поиска. Этой логикой поиска обычно является запрос к базе данных.

Рис. 13.1. Пользовательский интерфейс, предлагаемый по умолчанию в Search API

Несмотря на простоту, форма поиска контента по умолчанию фактически способна выполнять запрос по всем видимым элементам контента ноды вашего сайта. Это значит, что с помощью данного интерфейса можно производить поиск в заголовке ноды, ее теле, дополнительных специальных атрибутах, комментариях и таксономических терминах.

Расширенная форма поиска

Расширенная форма поиска (рис. 13.2) является еще одним способом фильтрации результатов поиска. Она расширяет базовую форму поиска, предоставляя возможность выбирать типы контента для ограничения поиска и получения простого интерфейса для ввода слов, фраз и слов, исключаемых из поиска.

Рис. 13.2. Опции расширенного поиска

Форма поиска по умолчанию может быть изменена реализацией хука поиска в модуле и последующим использованием `hook_form_alter()` для идентификатора формы `search_form` (см. главу 11) с целью предоставления интерфейса пользователю. Для формы на рис. 13.2 сделано то и другое. Модуль ноды реализует хук поиска, чтобы сделать ноды доступными для поиска (см. функции `node_search` в `modules/node/node.module`), и расширяет форму соответствующим интерфейсом (см. `node_form_search_form_alter()` в `modules/node/node.module`).

Дополнение к форме поиска

Давайте рассмотрим пример. Предположим, что мы используем `path.module` и хотим разрешить поиск псевдонимов URL на нашем сайте. Напишем небольшой модуль, который реализует `hook_search()` для того, чтобы сделать псевдонимы доступными для поиска, и предоставит дополнительную вкладку в интерфейсе поиска Drupal.

Добавление хуков поиска

Существует несколько функций `hook_search`, которые ваш модуль может использовать в Drupal 7.

- `hook_search_info()`. Эта функция позволяет вашему модулю сообщить модулю поиска о желании выполнять поиск по определенному в нем содержимому (например, в специальных типах нод, пользователях или комментариях), когда запускается поиск на сайте. Значения, устанавливаемые в этой функции, определяют вкладку, отображаемую в верхней части формы поиска для типа контента, поиск которого осуществляет ваш модуль (т.е. Content (Контент), Users (Пользователи), Comments (Комментарии)), и значение пути, добавляемое к URL после `/search` (например, `/search/node`).
- `hook_search_execute($keys = NULL)`. Эта функция выполняет поиск набора ключевых слов, которые введены пользователем и переданы в функцию в виде строки.
- `hook_search_reset()`. Эта функция вызывается, когда индекс поиска должен быть перестроен. Она используется модулями, которые также реализуют `hook_update_index()`. Если ваш модуль отслеживает размер своего индексируемого контента, с помощью этой функции можно сбросить счетчики модуля при подготовке к переиндексации.
- `hook_search_status()`. Эта функция сообщает состояние переиндексации контента в базе данных. Она возвращает значение общего количества элементов, подлежащих индексации, и число элементов, которые осталось проиндексировать.
- `hook_search_access()`. Эта функция позволяет модулю определить разрешения на доступ к вкладке поиска. Если пользователь не обладает соответствующими разрешениями, эта вкладка не будет для него отображаться на форме поиска.
- `hook_search_admin()`. Эта функция добавляет элементы к форме настройки параметров поиска.

Форматирование результатов поиска с помощью `hook_search_page()`

Если необходимо построить модуль, который предоставляет результаты поиска, вы можете взять на себя управление внешним видом страницы результатов, реализовав `hook_search_page()`. Если вы не реализуете этот хук, то результаты будут форматированы вызовом `theme_search_results($variables)`, у которого есть собственная реализация по умолчанию в `modules/search/search-results.tpl.php`. Не путайте эту функцию с `theme_search_results($variables)`, которая форматирует одиночный результат поиска и имеет реализацию по умолчанию в `modules/search/search-result.tpl.php`.

Обеспечение доступности для поиска псевдонимов путей

Давайте приступим к написанию примера. Мы создадим опцию поиска, которая позволит посетителям искать пути, реализовав множество хуков поиска.

На заметку! Для работы следующего примера понадобится активировать модуль `path` и назначить нодам некоторые пути (чтобы было чего искать). Перед тестированием примера придется также перестроить данные поискового индекса. Для этого понадобится выбрать `Administer`⇒`Site configuration`⇒`Search settings` (Администрирование⇒Конфигурация сайта⇒Параметры поиска), щелкнуть на кнопке `Re-index site` (Переиндексировать сайт) и затем посетить страницу `Administer`⇒`Reports`⇒`Status report` (Администрирование⇒Отчеты⇒Отчет о состоянии), чтобы запустить `cron` вручную. После запуска `cron` модуль `search` будет проиндексирован.

Создайте новую папку по имени `pathfinder` в каталоге `sites/all/modules/custom` и поместите в нее файлы с кодом, который приведен в листингах 13.1 и 13.2.

Листинг 13.1. Содержимое `pathfinder.info`

```
name = Pathfinder
description = Gives administrators the ability to search URL aliases.
package = Pro Drupal Development
core = 7.x
dependencies[] = path
files[] = pathfinder.module
```

Листинг 13.2. Содержимое `pathfinder.module`

```
<?php
/**
 * @file
 * Интерфейс поиска для псевдонимов URL.
 */
```

Оставьте файл `pathfinder.module` открытым в текстовом редакторе — вам еще предстоит работать с ним. Следующая функция, которую предстоит реализовать — это `hook_search_info()`. Данный хук помещает в верхнюю часть формы поиска вкладку для поиска псевдонимов URL:

```
/**
 * Реализация hook_search_info()
 */
function pathfinder_search_info() {
  return array(
    'title' => 'URL Aliases',
  );
}
```

Следующая функция проверяет наличие необходимых разрешений у посетителя для поиска псевдонимов URL:

```
/**
 * Реализация hook_search_access().
 */
function pathfinder_search_access() {
  return user_access('administer url aliases');
}
```

И, наконец, функция `hook_search_execute()` используется для выполнения поиска и возврата результатов:

```
/**
 * Реализация hook_search_execute().
 */
function pathfinder_search_execute($keys = NULL) {
  $find = array();
  $query = db_select('url_alias')->extend('PagerDefault');
  $query->fields('url_alias', array('source', 'alias'));
  $query->condition('alias', '%' . db_like($keys) . '%', 'LIKE');
  $result = $query
    ->limit(15)
    ->execute();
  foreach ($result as $alias) {
    $find[] = array('title' => $alias->alias,
      'link' => url($alias->source, array('absolute' => TRUE)));
  }
  return $find;
}
```

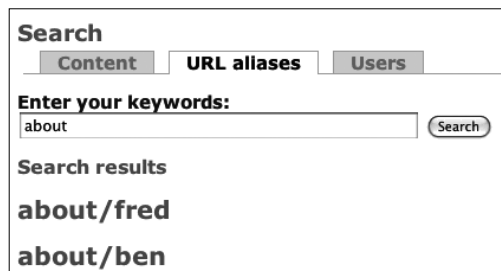
Когда Search API вызывает `hook_search_info()`, производится поиск имени вкладки меню, которая должна отображаться на обобщенной странице поиска (рис. 13.3). В нашем случае возвращается URL aliases (Псевдонимы URL). Возвращая имя вкладки меню, Search API передает ссылку на вкладку меню новой форме поиска.



The screenshot shows a search interface with the title "Search". Below the title are three tabs: "Content", "URL aliases" (which is selected and highlighted), and "Users". Under the tabs is a label "Enter your keywords:" followed by a text input field and a "Search" button.

Рис. 13.3. Форма поиска становится доступной благодаря возврату имени вкладки меню из `hook_search()`

Хук `hook_search_execute()` — это основная рабочая часть `hook_search()`. Он вызывается при отправке формы поиска, и его задача — собрать и вернуть результаты поиска. В предыдущем коде делался запрос к таблице `url_alias` с использованием условий поиска, отправленных посредством формы. Затем результаты запроса были собраны и отправлены обратно в виде массива. Результаты форматируются модулем поиска и отображаются пользователю, как показано на рис. 13.4.



The screenshot shows the same search interface as in Figure 13.3, but now it displays search results. The text input field contains the word "about". Below the input field, the text "Search results" is displayed, followed by two lines of results: "about/fred" and "about/ben".

Рис. 13.4. Результаты поиска, сформатированные модулем поиска

Использование HTML-индексатора поиска

До сих пор мы рассматривали взаимодействие с формой поиска по умолчанию, предоставляя простую реализацию `hook_search_execute()`. Однако когда мы переходим от поиска в простом столбце `VARCHAR` базы данных по маске `LIKE` к серьезно индексируемому контенту веб-сайта, эту задачу следует перепоручить встроенному в Drupal индексатору HTML.

Задача индексатора — эффективный поиск крупных порций HTML. Он делает это, обрабатывая контент, когда вызывается `cron` (через <http://example.com/cron.php>). Следовательно, имеется запаздывание, определяемое промежутком времени между тем, когда новый контент становится доступным для поиска, и тем, когда по расписанию запускается `cron`. Индексатор анализирует данные и разбивает текст на слова (этот процесс называется *разбиением на лексемы* (tokenization)), назначая оценку каждой лексеме на основе набора правил, который можно расширить с помощью Search API. Индексатор затем сохраняет эту информацию в базе данных и, когда поступает запрос на поиск, использует эти индексируемые таблицы вместо самих таблиц нод.

На заметку! При наличии сильно нагруженного сайта Drupal, к которому добавляются сотни новых нод между запусками `cron`, возможно, имеет смысл обратиться к некоторому внешнему по отношению к Drupal поисковому решению, такому как Solr (<http://drupal.org/project/apachesolr>).

Когда используется индексатор

В общем случае индексаторы используются при реализации поисковых механизмов, которые не ограничиваются стандартным подходом “наибольшего количества соответствующих слов”. Когда контент пропускается через (обычно сложный) набор правил, чтобы можно было провести упорядочивание в пределах индекса, говорят о *релевантности* поиска.

Если возникнет необходимость проведения поиска в объемном HTML-контенте, понадобится вся мощь индексатора. Одно из самых больших преимуществ Drupal состоит в том, что блоги, форумы, страницы и т.п. являются нодами. Их базовые структуры данных идентичны, и они, как правило, разделяют общую функциональность. Одна такая общая возможность связана с тем, что все ноды автоматически индексируются, если включен модуль поиска; никакого дополнительного программирования не требуется. Даже если вы создаете специальный тип ноды, поиск в ее контенте уже встроен, обеспечивая отображение произведенных модификаций во время визуализации.

Как работает индексатор

Индексатор имеет режим предварительной обработки, при котором текст проходит через набор правил для назначения оценок (score). Такие правила применяются по отношению к аббревиатурам, URL и числовым данным. В ходе фазы предварительной обработки другие модули имеют шанс добавить свою логику к этому процессу, чтобы выполнить собственные манипуляции с данными. Это пригодится при настройке под определенные языки, как показано здесь с использованием дополнительного модуля Porter-Stemmer (стеммер Портера):

- `resumé`⇒`resume` (снято ударение)
- `skipping`⇒`skip` (морфологический поиск)
- `skips`⇒`skip` (морфологический поиск)

Другим примером такой предварительной обработки языка является разбиение на слова для китайского, японского и корейского языков, чтобы гарантировать правильное индексирование текста, записанного иероглифами.

Совет. Модуль Porter-Stemmer (<http://drupal.org/project/porterstemmer>) — это пример модуля, который обеспечивает морфологический поиск (стемминг) слов для повышения эффективности поиска документов на английском языке. Аналогично, модуль разбиения китайских слов (Chinese Word Splitter) (<http://drupal.org/project/csplitter>) является расширенным препроцессором для улучшения поиска документов на китайском, японском и корейском языках. Упрощенный механизм разбиения китайских иероглифов на слова включен в модуль поиска и может быть найден на странице настройки параметров поиска.

После фазы предварительной обработки индексатор использует HTML-дескрипторы, чтобы найти наиболее важные слова (называемые лексемами), и назначает им уточненные оценки, основанные на оценках по умолчанию HTML-дескрипторов и количестве вхождений каждой лексемы. Эти оценки будут использоваться для определения окончательной релевантности лексемы. Вот полный список HTML-дескрипторов с их оценками по умолчанию (они определены в `search_index()`):

```
'h1' => 25,
'h2' => 18,
'h3' => 15,
'h4' => 12,
'h5' => 9,
'h6' => 6,
'u' => 3,
'b' => 3,
'i' => 3,
'strong' => 3,
'em' => 3,
'a' => 10
```

Давайте возьмем порцию HTML-кода и прогоним ее через индексатор, чтобы лучше понять, как он работает. На рис. 13.5 представлен краткий обзор того, как HTML-индексатор анализирует контент, назначая лексемам оценки и сохраняя эту информацию в базе данных.

Когда индексатор сталкивается с числовыми данными, разделенными знаками пунктуации, последние удаляются, а индексируются только числа. Благодаря этому упрощается поиск таких элементов, как даты, номера версий и IP-адреса. Процесс в середине рис. 13.5 показывает, как обрабатывается лексема слова, когда она не заключена между HTML-дескрипторами. Такие лексемы имеют вес 1. Последняя строка показывает контент, который заключен между дескрипторами выделения (`<em>`). Формула для определения общей оценки лексемы такова:

Количество совпадений × Вес HTML-дескриптора

Следует отметить, что Drupal индексирует также и отфильтрованный вывод нод. Например, если имеется набор фильтров ввода, предназначенный для автоматического преобразования URL в гиперссылки, или другой фильтр, служащий для преобразования разрывов строк в HTML-дескрипторы, индексатор видит этот контент со всей разметкой и при назначении оценок может соответственно учесть дескрипторы разметки. Большее влияние от индексации отфильтрованного вывода заметно на ноде, использующей фильтр с анализатором PHP для генерации динамического контента.

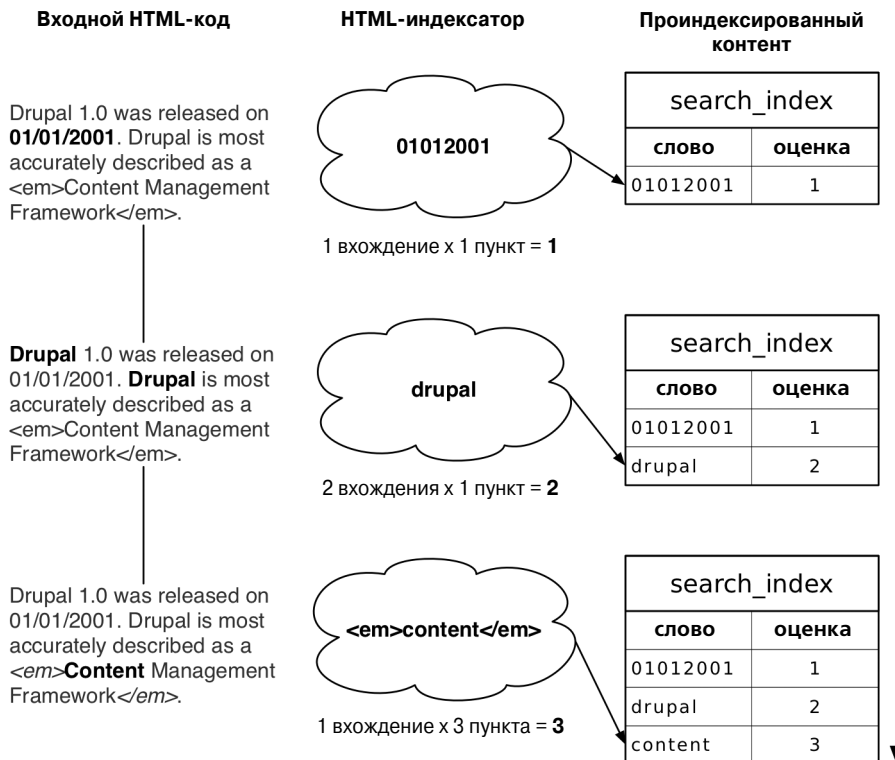


Рис. 13.5. Индексация порции HTML-кода и назначение оценок лексемам

Индексация динамического контента может оказаться реальной проблемой, но, поскольку индексатор Drupal видит только вывод, сгенерированный кодом PHP, динамический контент автоматически будет полностью доступным для поиска.

На заметку! Если контент подвергается изменениям, индекс не обновляется постоянно. Вместо этого индекс будет содержать динамический контент, который отображался, когда эта нота индексировалась по запуску `cron`. Затем он “замораживается” и не индексируется снова, пока не будут предприняты специальные шаги.

Когда индексатор встречает внутренние ссылки, то обрабатывает их особым образом. Если ссылка указывает на другую ноду, слова ссылки добавляются к контенту целевой ноды, благодаря чему упрощаются ответы на общие вопросы и становится проще найти релевантную информацию. Существуют два способа внедрения в работу индексатора.

- `hook_node_update_index($node)`. Можно добавить данные к ноду, которая в противном случае невидима, чтобы повысить релевантность поиска. Этот механизм в действии можно увидеть в ядре Drupal для терминов таксономии и комментариев, которые формально не являются частью ноды, но должны влиять на результаты поиска. Модуль комментариев (`Comment`) также реализует этот хук. Однако здесь кроется одна хитрость. Данный модуль использует функцию `comment_update_index` для установки предела по количеству комментариев, которые должны быть проиндексированы. Это представляет собой небольшой трюк с вмешательством в API-интерфейс.

- `hook_update_index()`. Можно использовать индексатор для индексирования HTML-контента, который не является частью ноды, с применением `hook_update_index()`. Реализация `hook_update_index()` ядра Drupal находится в `node_update_index()` из `modules/node/node.module`.

Оба эти хука вызываются во время запуска `cron` для индексации новых данных. На рис. 13.6 показан порядок, в котором выполняются названные хуки.

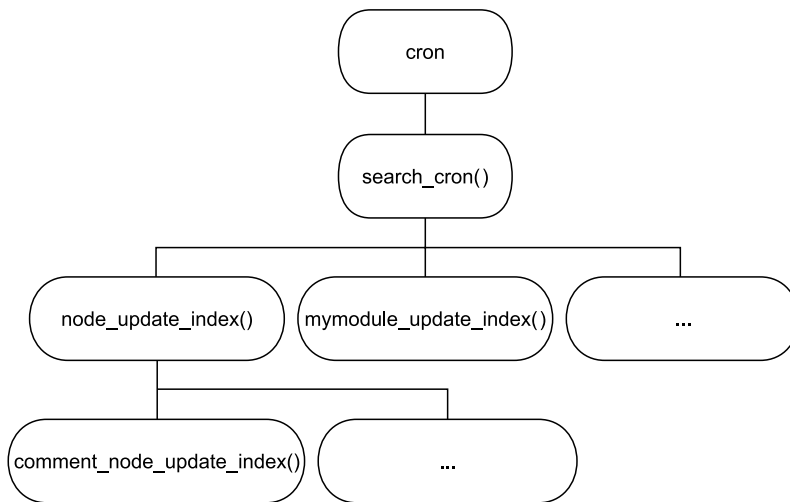


Рис. 13.6. Краткий обзор хуков индексации HTML

Все упомянутые хуки более подробно рассматриваются в последующих разделах.

Добавление метаданных к нодам: `hook_node_update_index()`

Когда среда Drupal индексирует ноду для обеспечения более быстрого поиска, она сначала запускает объект ноды с помощью `node_view()`. Модули могут решать, как данные будут отображаться, указывая, является ли контент подлежащим индексации. Например, предположим, что имеется нода с идентификатором 26. Части ноды, которые являются видимыми при просмотре URL `http://example.com/?q=node/26` — это то, что видит также и индексатор.

А что, если мы имеем дело со специальным типом ноды, содержащим скрытые данные, которые должны влиять на результаты поиска? Хорошим примером является модель книги `book.module`. Можно было бы проиндексировать названия глав наряду с каждой дочерней страницей, чтобы увеличить релевантность этих дочерних страниц.

```

/**
 * Реализация hook_node_update_index().
 */
function book_boost_node_update_index($node) {
  // Ноды книг имеют атрибут идентификатора ссылки на родителя.
  // Если он не равен нулю, мы можем заставить систему меню
  // извлечь родительский элемент меню, который даст заголовок.
  if ($node->type == 'book' && $node->book['plid']) {
    $item = menu_link_load($node->book['plid']);
    return '<h2>'. $item['title'] . '</h2>';
  }
}

```

Обратите внимание на то, что мы заключили заголовок в HTML-дескрипторы заголовка (`<h2>`), чтобы сообщить индексатору о более высоком относительном значении оценки этого текста.

На заметку! Хук `hook_node_update_index` предназначен для добавления метаданных только к нодам. Чтобы индексировать элементы, не являющиеся нодами, используйте хук `hook_update_index()`.

Индексация контента, не относящегося к нодам: `hook_update_index()`

Когда нужно представить поисковому механизму контент, который не состоит из нод Drupal, можно внедриться непосредственно в индексатор и выдать ему любые текстовые данные, подобным образом сделав их доступными для поиска внутри Drupal. Предположим, что ваша группа занимается поддержкой унаследованного приложения, которое использовалось для ввода и просмотра комментариев технического характера о продуктах в течение нескольких прошедших лет. По веским причинам вы не можете заменить это приложение решением Drupal, но хотели бы иметь возможность проводить поиск в комментариях внутри Drupal. Никаких проблем. Предположим, что унаследованное приложение сохраняет информацию в таблице базы данных по имени `technote`. Мы создадим короткий модуль, который будет посылать информацию из этой базы данных в индексатор Drupal, используя `hook_update_index()`, и представлять результаты поиска с помощью хуков поиска.

На заметку! Если нужно индексировать контент, хранящийся в базах данных, отличных от Drupal, обратитесь к главе 5 для получения дополнительной информации о подключении к множеству баз данных.

Создайте папку по имени `legacysearch` в каталоге `sites/all/modules/custom`. Если необходимо подготовить унаследованную базу данных для целей тестирования, создайте файл по имени `legacysearch.install` и поместите в него следующий код:

```
<?php

/**
 * Реализация hook_install().
 */
function legacysearch_install() {
  $fields = array('id' => 1, 'title' => 'Web 1.0 Emulator', 'note' =>
'<p>This handy product lets you emulate the blink tag but in hardware...a
perfect gift.</p>', 'last_modified' => 1172502517);
  db_insert('technote')
    ->fields($fields)
    ->execute();

  $fields = array('id' => 2, 'title' => 'Squishy Debugger', 'note' =>
'<p>Fully functional debugger inside a squishy gel case. The embedded ARM
processor heats up...</p>', 'last_modified' => 1172502517);
  db_insert('technote')
    ->fields($fields)
    ->execute();
}
/**
 * Реализация hook_uninstall().
 */
```



```

function legacysearch_uninstall() {
    drupal_uninstall_schema('legacysearch');
}

/**
 * Реализация hook_schema().
 */
function legacysearch_schema() {
    $schema['technote'] = array(
        'description' => t('A database with some example records.'),
        'fields' => array(
            'id' => array(
                'type' => 'serial',
                'not null' => TRUE,
                'description' => t("The tech note's primary ID."),
            ),
            'title' => array(
                'type' => 'varchar',
                'length' => 255,
                'description' => t("The tech note's title."),
            ),
            'note' => array(
                'type' => 'text',
                'description' => t('Actual text of tech note.'),
            ),
            'last_modified' => array(
                'type' => 'int',
                'unsigned' => TRUE,
                'description' => t('Unix timestamp of last modification.'),
            ),
        ),
        'primary key' => array('id'),
    );
    return $schema;
}

```

Этот модуль обычно не нуждается в таком инсталляционном файле, поскольку унаследованная база данных уже существует. Мы используем его только для целей тестирования. Взамен можно откорректировать запросы внутри модуля, чтобы подключиться к существующей таблице (отличной от Drupal). В следующих запросах предполагается, что данные хранятся в отличной от Drupal базе данных с подключением, определенным в массиве \$databases внутри settings.php.

Теперь создайте файл sites/all/modules/custom/legacysearch/legacysearch.info со следующим кодом:

```

name = Legacy Search
description = Enables searching of external content within Drupal.
package = Pro Drupal Development
core = 7.x
files[] = legacysearch.install
files[] = legacysearch.module

```

И, наконец, создайте файл sites/all/modules/custom/legacysearch/legacysearch.module со следующим кодом:

```
<?php
/**
 * @file
 * Позволяет проводить поиск в контенте, отличном от Drupal.
 */
```

Оставьте `legacysearch.module` открытым в текстовом редакторе. Мы добавим хук `hook_update_index()`, который передает унаследованные данные индексатору HTML. После создания этих файлов можно активировать модуль. Также понадобится перейти в `admin/config/search/settings` и включить `legacy_search` как один из активных модулей поиска, а после сохранения щелкнуть на кнопке **Re-index site** (Переиндексировать сайт), чтобы перестроить индексы, включая `legacysearch`.

```
/**
 * Реализация hook_search_info().
 */
function legacysearch_search_info() {
  return array(
    'title' => 'Tech Notes',
  );
}

/**
 * Реализация hook_search_reset().
 */
function legacysearch_search_reset() {
  variable_del('legacysearch_cron_last');
  variable_del('legacysearch_cron_last_id');
  return;
}

/**
 * Функция завершения, гарантирующая запоминание последнего обработанного элемента
 */
function legacysearch_update_shutdown() {
  global $last_change, $last_id;
  if ($last_change && $last_id) {
    variable_set('legacysearch_cron_last', $last_change);
    variable_set('legacysearch_cron_last_id', $last_id);
  }
}

/**
 * Реализация hook_update_index().
 */
function legacysearch_update_index() {
  global $last_change, $last_id;
  register_shutdown_function('legacysearch_update_shutdown');

  $last_id = variable_get('legacysearch_cron_last_id', 0);
  $last_change = variable_get('legacysearch_cron_last_change', 0);

  db_set_active('legacy');
  $result = db_query("SELECT id, title, note, last_modified FROM {technote}
WHERE id > :last_id OR last_modified > :last_change",
    array(':last_id' => $last_id, ':last_change' => $last_change));
  db_set_active('default');
```

```

foreach($result as $data) {
    $last_change = $data->last_modified;
    $last_id = $data->id;
    $text = '<h1>' . check_plain($data->title) . '</h1>' . $data->note;
    search_index($data->id, 'technote', $text);
    variable_set('legacysearch_cron_last', $data->last_modified);
    variable_set('legacysearch_cron_last_id', $data->id);
}
}

/**
 * Реализация hook_search_execute().
 */
function legacysearch_search_execute($keys = NULL) {

    // Установить URL для внедрения в ссылку, на которой пользователь может
    // щелкнуть, чтобы получить доступ к унаследованному сайту.
    $legacy_url = 'http://technotes.example.com';

    // Настроить и запустить запрос.
    $query = db_select('search_index',
        'i')->extend('SearchQuery')->extend('PagerDefault');
    $query->join('technote', 't', 't.id = i.sid');
    $query->searchExpression($keys, 'technote');

    // Если результаты отсутствуют, вернуть пустой результирующий набор.
    if (!$query->executeFirstPass()) {
        return array();
    }

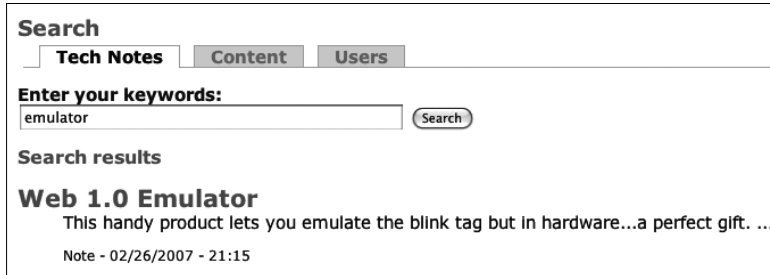
    // Если первый проход вернул хотя бы одну запись, запустить поиск.
    $found = $query
        ->limit(10)
        ->execute();

    // Создать вывод с результатами поиска.
    foreach ($found as $item) {
        // Первым делом извлечь значения из унаследованной таблицы
        // для отображения в результатах поиска.
        db_set_active('legacy');
        $note = db_query("SELECT * FROM {technote} where id = :sid",
            array(':sid' => $item->sid));
        db_set_active('default');

        // Форматировать результаты поиска.
        $results[] = array(
            'link' => url($legacy_url . 'note.pl', array('query' => $item->sid,
                'absolute' => TRUE)),
            'type' => t('Note'),
            'title' => $note->title,
            'date' => $note->last_modified,
            'score' => $item->score,
            'snippet' => search_excerpt($keys, $note->note));
    }
    return $results;
}

```

После запуска `cron` и индексирования информации комментарии технического характера станут доступными для поиска (рис. 13.7). Они будут проиндексированы внутри Drupal, но `legacysearch_search()` вернет результаты поиска, построенные из (и указывающие на) унаследованной системы.



The screenshot shows a search interface with a header 'Search' and three tabs: 'Tech Notes', 'Content', and 'Users'. The 'Tech Notes' tab is selected. Below the tabs is a text input field with the text 'emulator' and a 'Search' button. Below the search bar, the text 'Search results' is displayed. The first result is titled 'Web 1.0 Emulator' and has a description: 'This handy product lets you emulate the blink tag but in hardware...a perfect gift. ...'. Below the description, the text 'Note - 02/26/2007 - 21:15' is displayed.

Рис. 13.7. Поиск во внешней унаследованной базе данных

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Настраивать форму поиска.
- Понимать, как работает хук поиска.
- Понимать, как работает индексатор HTML.
- Вмешиваться в работу индексатора для контента любого вида.

ГЛАВА 14

Работа с файлами

Система Drupal обеспечивает загрузку и выгрузку файлов множеством способов. В этой главе мы рассмотрим общедоступные и приватные файлы и то, как они обслуживаются, кратко упомянем об обработке медиа-файлов и опишем хук аутентификации файлов Drupal.

Как Drupal обслуживает файлы

Давным-давно, где-то в укромном месте, было решено скрывать структуру файлов от использующих их приложений. Эта система получила название “потoki” и стала краеугольным камнем системы Unix, в настоящее время развившейся до системы Linux, которая приводит в движение все наши веб-серверы. Идея состояла в том, чтобы пользователь мог применять один набор функций (системные вызовы) для взаимодействия с любым видом файловой системы (или чем-либо, не являющимся файлами) и при этом не замечать никаких различий. Идея, названная “потокami”, нашла свое отражение практически во всех современных вычислительных технологиях, а в PHP была дополнена идеей именования файлов с использованием нотации “потокowego упаковщика”. Это означает всего лишь, что имя файла содержит “схему”, за которой следуют символы `://` и затем “цель”, которая по существу является путем в файловой системе (при условии, что речь идет о файловой системе).

Для имен файлов в Drupal используется нотация потокowego упаковщика. Для общедоступных файлов запись принимала бы вид `public://что-либо...`, для приватных — `private://что-либо...`. Этот же подход приемлем также и для временных файлов. Кроме того, модуль Drupal может реализовать класс `DrupalStreamWrapperInterface`, чтобы предоставить дополнительные упаковщики потоков.

Модуль `File Example` предлагает упаковщик `session://`, который в качестве демонстрации позволяет реализовать хранилище файлов в `$_SESSION`. (Хочется надеяться, что это хранилище не найдет никакого законного применения.) PHP также предоставляет ряд других типов схем, в том числе `file://`, `ftp://`, `http://` и т.д., все они могут использоваться внутри модуля.

Управляемый и неуправляемый API-интерфейс Drupal

Существует два “уровня” API-интерфейсов работы с файлами Drupal — “управляемый” и “неуправляемый”. (Естественно, то же самое касается и файловых функций PHP.) “Управляемые” API-интерфейсы поддерживают запись в таблице `file_managed` так, чтобы к файлу можно было получить доступ позже. Большинство модулей, которые ис-

пользуют файл при решении более чем непосредственной одноразовой задачи, будут работать с управляемыми файлами.

Неуправляемые функции выполняют в основном то же, что и их базовые PHP-аналоги. Если вы просите, чтобы функция `file_unmanaged_copy()` скопировала файл, она делает это, проверяя при этом полномочия и ряд других условий. Ничто не помещается в базу данных.

Внимание! Поскольку общедоступный и приватный методы хранения файлов приводят к генерации различных URL для загрузки, прежде чем начать выгрузку файлов, важно выбрать опцию, которая наиболее подходит для данного сайта, и придерживаться выбранного метода.

Чтобы установить пути файловой системы и указать, какой метод загрузки должен использоваться, перейдите к странице Configuration⇒File system (Конфигурация⇒Файловая система), показанной на рис. 14.1.

File system

Dashboard » Configuration » Media

Public file system path

A local file system path where public files will be stored. This directory must exist and be writable by Drupal. This directory must be relative to the Drupal installation directory and be accessible over the web.

Private file system path

A local file system path where private files will be stored. This directory must exist and be writable by Drupal. This directory should not be accessible over the web.

Temporary directory

A local file system path where temporary files will be stored. This directory should not be accessible over the web.

Default download method

☒ Public local files served by the webserver.

This setting is used as the preferred download method. The use of public files is more efficient, but does not provide any access control.

Рис. 14.1. Интерфейс для установки в Drupal настроек, относящихся к файлам. Каталог, указанный в пути общедоступной и приватной файловой системы, должен быть создан, и ему должны быть назначены соответствующие полномочия

Общедоступные файлы

Самая простая конфигурация — метод загрузки общедоступных файлов, при использовании которого Drupal не участвует в процессе загрузки. При выгрузке файлов Drupal просто сохраняет их в каталоге, который был указан на странице Configuration⇒File system, и отслеживает URL файлов в таблице базы данных (благодаря чему Drupal известно, какие файлы доступны, кто их выгрузил и т.д.). При запросе файла он передается непосредственно веб-сервером по протоколу HTTP как статический файл, и Drupal вообще не участвует в этом процессе. Преимущество этого метода — скорость, потому что

никакого PHP-кода выполняться не должно. Однако при этом не проверяются и разрешения Drupal для пользователя.

При указании пути файловой системы должна существовать соответствующая папка, доступная для записи PHP-кодом. Обычно пользователь (операционной системы), от имени которого запущен веб-сервер, совпадает с пользователем, от имени которого выполняется PHP-код. Таким образом, предоставление этому пользователю права на запись в папку `files` означает разрешение на выгрузку файлов для Drupal. Сделав это, не забудьте указать путь файловой системы в меню Configuration⇒File system. Как только эти изменения будут сохранены, Drupal автоматически создаст файл `.htaccess` в папке `files`. Это необходимо для защиты сервера от известной брешки в системе безопасности Apache, которая позволяет пользователям выгружать и запускать сценарии, содержащиеся в выгруженных файлах (см. <http://drupal.org/node/66763>). Удостоверьтесь, что папка `files` содержит файл `.htaccess` со следующей информацией:

```
SetHandler Drupal_Security_Do_Not_Remove_See_SA_2006_006
Options None
Options +FollowSymLinks
```

Совет. При выполнении Drupal в кластере веб-серверов каталог временных файлов должен разделяться между всеми веб-серверами. Поскольку Drupal может использовать один запрос для выгрузки файла, а второй для изменения его статуса с временного на постоянный, многие схемы балансировки загрузки будут приводить к тому, что временный файл попадет на один сервер, а второй запрос — на другой. Когда такое происходит, будет казаться, что файлы выгружаются должным образом, но они никогда не будут появляться в нодах или в контенте, к которому они прикреплены. Удостоверьтесь, что все веб-серверы используют один и тот же каталог для хранения временных файлов, и применяйте балансировку загрузки на основе сеансов. Ваш каталог `files`, как и база данных, должен быть глобальным для веб-серверов.

Приватные файлы

В режиме приватной загрузки папка `files` может быть расположена где угодно, среда PHP должна иметь возможность чтения и записи в ней, и эта папка может не быть (в большинстве случаев это условие обязательно) непосредственно доступной веб-серверу.

Безопасность приватных файлов обеспечивается ценой снижения производительности. Вместо того чтобы делегировать работу по обслуживанию файла веб-серверу, Drupal берет на себя ответственность за проверку разрешений на доступ и обслуживание файлов, и полностью перезагружается при каждом запросе файла.

Параметры настройки PHP

Некоторые настройки в `php.ini` легко упустить из виду, однако они важны для выгрузок файлов. Первой из них является `post_max_size` из раздела Data Handling (Обработка данных) в `php.ini`. Поскольку файлы выгружаются по HTTP-запросу POST, попытки выгрузки файлов с размером, который превышает `post_max_size`, будут терпеть неудачу из-за ограничения на объем отправляемых POST-данных.

```
; Максимальный размер данных POST, который будет воспринимать PHP.
post_max_size = 8M
```

Раздел File Uploads (Выгрузки файлов) файла `php.ini` содержит несколько более важных параметров настройки. Здесь можно определить, разрешена ли выгрузка файлов, и каким должен быть максимальный размер выгружаемых файлов.

```

//////////
; Выгрузки файлов ;
//////////
; Разрешены ли загрузки файлов по протоколу HTTP.
file_uploads = On

; Временный каталог для файлов, выгруженных по протоколу HTTP (если
; не указан, используется каталог, установленный в системе по умолчанию).
;upload_tmp_dir =

; Максимальный разрешенный размер для выгруженных по HTTP файлов.
upload_max_filesize = 20M

```

Если загрузка файлов терпит неудачу, проверьте корректность этих настроек. Также обратите внимание, что значение `upload_max_filesize` должно быть меньше, чем `post_max_size`, а оно, в свою очередь, должно быть меньше, чем `memory_limit`:

```
upload_max_filesize < post_max_size < memory_limit
```

Две последние настройки, о которых следует помнить — это `max_execution_time` и `max_input_time`. Если сценарий превысит эти ограничения при загрузке файла, то PHP прекратит его выполнение. Проверьте эти настройки, если окажется, что загрузки по медленным Интернет-соединениям терпят неудачу.

```

//////////
; Ограничения ресурсов ;
//////////
max_execution_time = 60 ; Максимальное время выполнения
                        ; для каждого сценария, в секундах.
                        ; xdebug использует его, поэтому при отладке
                        ; установите его очень большим.
max_input_time = 60    ; Максимальное время, которое каждый
                        ; сценарий может тратить
                        ; на синтаксический разбор данных запроса.

```

Во время отладки `max_execution_time` понадобится установить в более высокое значение (например, 1600), чтобы отладчик не приводил к возникновению тайм-аута. Однако имейте в виду, что при высоком уровне загрузки сервера процессы Apache могут оказаться связанными на длительное время, пока выгружаются файлы, создавая потенциальные проблемы масштабируемости.

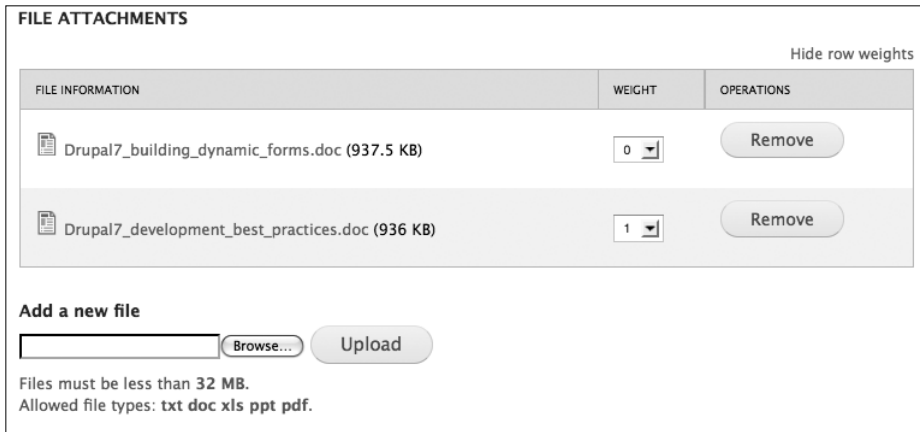
Обработка медиа-данных

API-интерфейс для работы с файлами (находящийся в `includes/file.inc`) не предоставляет обобщенный пользовательский интерфейс загрузки файлов. Чтобы заполнить этот пробел для большинства конечных пользователей, API-интерфейс работы с полями (см. главу 8) предоставляет функциональные возможности, позволяющие удовлетворить большинство требований загрузки файлов.

Поле загрузки

Чтобы активировать загрузку файлов типа контента, вначале к заданному типу контента нужно добавить поле для загрузки файлов. Для этого перейдите к странице Structure⇒Content Types (Структура⇒Типы контента). На странице Content types щелкните на ссылке `manage fields` (управление полями) того типа контента, к которому нужно добавить возможность загрузки файлов, и добавьте новое поле файла к типу кон-

тента. Немедленно после добавления к типу контента поле выгрузки файла появится на экране редактирования контента для данного типа контента (рис. 14.2).



FILE INFORMATION	WEIGHT	OPERATIONS
 Drupal7_building_dynamic_forms.doc (937.5 KB)	0	<button>Remove</button>
 Drupal7_development_best_practices.doc (936 KB)	1	<button>Remove</button>

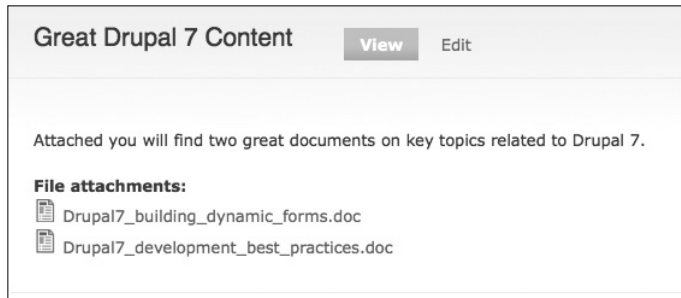
Add a new file

Browse... Upload

Files must be less than 32 MB.
Allowed file types: txt doc xls ppt pdf.

Рис. 14.2. Поле File attachments, добавленное к форме ноды

После того как файл был выгружен в форме редактирования ноды, модуль `upload.module` может добавить ссылки загрузки к выгружаемым файлам под телом ноды. Ссылки видимы тем, кто обладает разрешением `view uploaded files` (просмотр выгруженных файлов), как показано на рис. 14.3.



Great Drupal 7 Content View Edit

Attached you will find two great documents on key topics related to Drupal 7.

File attachments:

-  Drupal7_building_dynamic_forms.doc
-  Drupal7_development_best_practices.doc

Рис. 14.3. Общее представление списка файлов, выгруженных в ноду

Это обобщенное решение вряд ли подойдет для большинства случаев, поэтому в следующем разделе мы рассмотрим несколько специфических примеров.

Видео и аудио

Множество модулей, помогающих управлять медиа-контентом, таким как видео-файлы, Flash-контент, слайд-шоу и т.д., можно найти по адресу www.drupal.org/project/modules.

API-интерфейс для работы с файлами

API-интерфейс для работы с файлами находится в `include/file.inc`. В этом разделе мы рассмотрим некоторые его часто используемые функции. Более подробную информацию можно найти в документации по API, которая доступна по адресу <http://api.drupal.org/api/group/file/7>.

Схема базы данных

Хотя система Drupal сохраняет файлы на диске, она все же использует базу данных для хранения множества метаданных, относящихся к файлам. В дополнение к информации об авторстве файла, MIME-типе и местоположении для выгруженных файлов поддерживается информация о версиях. Схема таблицы `file_managed` приведена в табл. 14.1.

Таблица 14.1. Таблица `file_managed`

Поле*	Тип	Значение по умолчанию	Описание
fid	serial		Первичный ключ
<i>uid</i>	int	0	Идентификатор пользователя, ассоциированного с файлом
<i>filename</i>	varchar(255)	' '	Имя файла
<i>uri</i>	varchar(255)	' '	Унифицированный идентификатор ресурса (URI) для доступа к файлу (локальному или удаленному)
<i>filemime</i>	varchar(255)	' '	MIME-тип файла
<i>filesize</i>	int	0	Размер файла в байтах
<i>status</i>	int	0	Флаг, указывающий, является файл временным (1) или постоянным (0)
<i>timestamp</i>	int	0	Временная метка Unix, указывающая время добавления файла

* Полу жирным выделен первичный ключ, а курсивом — индексированные поля.

Управление механизмом связи выгруженных файлов с соответствующим контентом осуществляется посредством таблицы `field_data_field_file_xxxxxx`, где `xxxxxx` представляет уникальное имя, назначенное полю данной формы при его добавлении к типу контента. Схема всех этих таблиц идентична и показана в табл. 14.2.

Таблица 14.2. Таблица `upload`, используемая модулем Upload

Поле*	Тип	Значение по умолчанию	Описание
etid	int	0	Идентификатор типа объекта, к которому присоединены эти данные
<i>bundle</i>	varchar		Группа экземпляров полей, к которой принадлежит данная строка
deleted	tinyint	0	Булевское значение, указывающее, был ли удален этот элемент данных
<i>entity_id</i>	int		Идентификатор объекта, к которому присоединены эти данные (например, идентификатор ноды)
<i>revision_id</i>	int	NULL	Идентификатор версии объекта, к которому присоединены эти данные

Поле*	Тип	Значение по умолчанию	Описание
language	varchar		Язык этого элемента данных
delta	int		Порядковый номер этого элемента данных
<i>field_XXXXXX_fid</i>	int	NULL	<i>file_managed.id</i> , на который осуществляется ссылка в этом поле, где XXXXXX заменяется именем поля из типа контента
field_XXXXXX_display	tinyint	1	Флаг, управляющий тем, должен ли этот файл отображаться при просмотре контента
Field XXXXXX_description	Text	NULL	Описание файла

* Полу жирным выделен первичный ключ, а курсивом — индексированные поля.

Общие задачи и функции

Если требуется что-то сделать с файлом, скорее всего, в API-интерфейсе работы с файлами найдется удобная функция, которую можно использовать. Рассмотрим некоторые из таких функций.

Выяснение заданного по умолчанию URI файлов

Функция `file_default_scheme()` возвращает схему по умолчанию (например, общедоступную или приватную) и может использоваться для определения URI места хранения этих файлов. Например, `file_default_scheme().":/"` представляет расположение по умолчанию места записи выгружаемых файлов.

Сохранение данных в файл

Иногда нужно просто сохранить данные в файл. Именно это и делает следующая функция:

```
file_save_data($data, $destination = NULL, $replace = FILE_EXISTS_RENAME)
```

Параметр `$data` станет содержимым файла. Параметр `$dest` — URI назначения. Параметр `$replace` определяет поведение Drupal в случае, если файл с указанным именем уже имеется в каталоге назначения. Возможные его значения перечислены в табл. 14.3.

Таблица 14.3. Константы, определяющие поведение Drupal, когда файл с тем же именем существует в каталоге назначения

Имя	Описание
FILE_EXISTS_REPLACE	Заменяет существующий файл текущим
FILE_EXISTS_RENAME	Добавляет к имени нового файла символ подчеркивания и номер, чтобы сделать его уникальным
FILE_EXISTS_ERROR	Прерывает работу и возвращает значение FALSE

Ниже приведен краткий пример помещения строки в файл, находящийся в каталоге файловой системы Drupal:

```
<?php
$filename = 'testfile.txt';
$dest = file_build_uri($filename);
file_save_data('My data', $dest, FILE_EXISTS_REPLACE);
```

Переменная `$dest` должна содержать допустимый URI упаковщика потока. В предыдущем примере используется функция `file_build_uri` для создания допустимого URI упаковщика потока, указывающего на целевой каталог, которым в этом случае является общедоступный каталог файлов по умолчанию.

Копирование и перемещение файлов

Следующие функции позволяют работать с файлами, которые уже существуют в файловой системе.

```
file_copy($source, $destination = NULL, $replace = FILE_EXISTS_RENAME)
```

Функция `file_copy()` копирует файлы в путь системных файлов Drupal (обычно `sites/default/files`). Параметр `$source` — это объект файла, `$destination` — строка, содержащая каталог назначения, в который нужно скопировать файл (в виде допустимого URI упаковщика потока), а `$replace` — действие, которое система Drupal должна предпринять, если файл уже существует в каталоге назначения.

```
file_move($source, $destination = NULL, $replace = FILE_EXISTS_RENAME)
```

Функция `file_move()` работает подобно `file_copy()` (на самом деле она вызывает функцию `file_copy()`, но также удаляет исходный файл посредством функции `file_delete()`).

Проверка каталогов

Функция `file_prepare_directory(&$directory, $options=FILE_MODIFY_PERMISSIONS)` проверяет, существует ли каталог и доступен ли он для записи, что целесообразно выполнить, прежде чем предпринимать попытку записи в этот каталог. Следующий пример проверяет, существует ли каталог `sites/default/files` и доступен ли он для записи.

```
<?php
$directory = 'sites/default/files';
if (file_prepare_directory(&$directory, $options = FILE_MODIFY_
PERMISSIONS)) {
    echo "The directory exists and is writeable";
    // Каталог существует и доступен для записи
} else {
    echo "The file does not exist or it is not writeable";
    // Файл не существует или не доступен для записи
}
```

Выгрузка файлов

Хотя API-интерфейс для работы с полями и его поле выгрузки предоставляет полноценную реализацию выгрузки файлов для нод, иногда может потребоваться просто выгрузить файл, не ассоциированный ни с одной нодой. Описанные ниже функции могут оказаться полезными в подобной ситуации.

```
file_save_upload($source, $validators = array(), $destination = FALSE,
$replace = FILE_EXISTS_RENAME)
```

Параметр `$source` — строка, которая определяет путь файла или URI выгруженного файла, который нужно сохранить. Параметр `$validators` — необязательный ассоциативный массив функций обратного вызова, используемых для проверки файла. Если этот массив функций проверки не указан, Drupal выполняет основную проверку того, что расширение файла — одно из `jpg jpeg gif png txt doc xls pdf ppt pps odt ods odp`. Параметр `$destination` — строка, которая содержит URI каталога, в который нужно копировать источник, а параметр `$replace` позволяет указать, должен ли выгруженный файл заменить существующий файл, быть переименован посредством добавления к имени файла увеличивающегося номера, либо выдавать ошибку. Ниже приведена функция проверки из пользовательского модуля, который выгружает картинку пользователя, используя функцию `file_save_upload`. Функция устанавливает три проверки: проверяет, является ли файл изображением, удостоверяется, что разрешение изображения равно 85×85, и проверяет размер файла. Само изображение поступает из поля выгрузки файла Upload picture (Выгрузить картинку) в пользовательской форме (рис. 14.4). Результирующий объект показан на рис. 14.5.

```
function user_validate_picture(&$form, &$form_state) {
  // При необходимости проверить выгружаемую картинку.
  $validators = array(
    'file_validate_is_image' => array(),
    'file_validate_image_resolution' =>
      array(variable_get('user_picture_dimensions', '85x85')),
    'file_validate_size' =>
      array(variable_get('user_picture_file_size', '30') * 1024),
  );
  // Сохранить файл в качестве временного.
  $file = file_save_upload('picture_upload', $validators);
  if ($file === FALSE) {
    form_set_error('picture_upload', t("Failed to upload the picture image;
    the %directory directory doesn't exist or is not writable.",
    array('%directory' => variable_get('user_picture_path', 'pictures'))));
  }
  elseif ($file !== NULL) {
    $form_state['values']['picture_upload'] = $file;
  }
}
```

Параметр `$dest` в функции `file_save_upload()` не обязателен и может содержать каталог, в который будет скопирован файл. Например, при обработке файлов, присоединенных к ноте, в качестве значения `$dest` модуль выгрузки использует функцию `file_directory_path()` (которая по умолчанию возвращает `sites/default/files`), как показано на рис. 14.6. Если значение `$dest` не указано, будет использоваться временный каталог.

Параметр `$replace` определяет, что должна делать система Drupal, когда файл с таким же именем уже существует. Возможные значения перечислены в табл. 14.3.

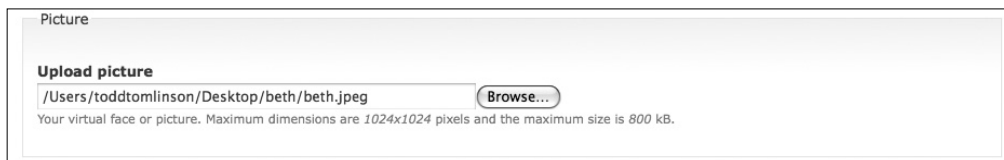


Рис. 14.4. Поле ввода файла для элемента формы `user_picture`, как оно выглядит на странице `My account` (Моя учетная запись)

```

Array
(
    [files] => Array
        (
            [name] => Array
                (
                    [picture_upload] => beth.jpeg
                )

            [type] => Array
                (
                    [picture_upload] => image/jpeg
                )

            [tmp_name] => Array
                (
                    [picture_upload] => /Applications/MAMP/tmp/php/phpS74bvc
                )

            [error] => Array
                (
                    [picture_upload] => 0
                )

            [size] => Array
                (
                    [picture_upload] => 176114
                )
        )
    )
)

```

Рис. 14.5. Результирующий объект после выполнения HTTP-запроса POST

```

stdClass Object
(
    [uid] => 1
    [status] => 0
    [filename] => beth.jpeg
    [uri] => temporary://beth.jpeg
    [filemime] => image/jpeg
    [filesize] => 176114
    [source] => picture_upload
    [destination] => temporary://beth.jpeg
    [timestamp] => 1281230004
    [fid] => 29
)

```

Рис. 14.6. Объект файла в том виде, как он передается функциям проверки `file_save_upload()`

Возвращаемое значение функции `file_save_upload()` представляет собой целиком заполненный объект файла (как показано на рис. 14.6) или 0, если что-то пошло не так.

После вызова `file_save_upload()` во временном каталоге Drupal появляется новый файл, а в таблице `files` базы данных — новая запись. Запись содержит те же значения, что и объект `file`, показанный на рис. 14.6.

Обратите внимание, что поле `status` установлено в 0. Это значит, что до тех пор, пока это зависит от Drupal, данный файл считается временным. Сделать его постоянным — задача вызывающего кода. Продолжая рассмотрение нашего примера выгрузки пользовательского изображения, мы видим, что модуль `user` применяет подход с копированием этого файла в каталог, определенный Drupal-переменной `user_picture_path`, и переименованием его с использованием идентификатора пользователя.

```
// Обработка выгрузок изображений.
if (!empty($edit['picture']->fid)) {
    $picture = $edit['picture'];

    // Если картинка представляет собой временный файл, переместить
    // его конечное расположение и сделать постоянным.
    if (($picture->status & FILE_STATUS_PERMANENT) == 0) {
        $info = image_get_info($picture->uri);
        $picture_directory = variable_get('file_default_scheme', 'public')
        . '://' . variable_get('user_picture_path', 'pictures');

        // Подготовить каталог изображений.
        file_prepare_directory($picture_directory, FILE_CREATE_DIRECTORY);
        $destination = file_stream_wrapper_uri_normalize($picture_directory
        . '/picture-' . $account->uid . '-' . REQUEST_TIME . '.' . $info['extension']);
        if ($picture = file_move($picture, $destination, FILE_EXISTS_RENAME))
        {
            $picture->status |= FILE_STATUS_PERMANENT;
            $edit['picture'] = file_save($picture);
        }
    }
}.
```

В результате выгруженное изображение будет перемещено в каталог `sites/default/files/pictures/directory` и файл станет постоянным.

Если параметр `$dest` был предоставлен, а файл перемещен в финальное место назначения вместо временного каталога, вызывающий код может изменить состояние записи в таблице `files` на постоянное, вызвав для этого `file_save($file)` с `$file`, установленным в значение полного объекта файла (как показано на рис. 14.7), а `status`, установленным в значение `FILE_STATUS_PERMANENT`. Согласно `includes/file.inc`, если вы планируете использовать дополнительные константы состояния в собственных модулях, установку их значений следует начинать с 256, поскольку значения 0, 1, 2, 4, 8, 16, 32, 64 и 128 зарезервированы для ядра.

Функции проверки, которые могут использоваться с `file_save_upload()`, описаны в последующих разделах.

file_validate_extensions(\$file, \$extensions)

Параметр `$file` — это объект файла. Параметр `$extensions` — строка, состоящая из разделенных пробелами расширений файла. Функция вернет пустой массив, если расширение файла разрешено, и массив сообщений об ошибках вроде `Only files with the following extensions are allowed: jpg jpeg gif png txt doc xls pdf ppt pps odt ods odp` (Разрешены файлы только со следующими расширениями: `jpg jpeg gif png txt doc xls pdf ppt pps odt ods odp`), если расширение файла запрещено. Эта функция также может служить средством проверки достоверности для `file_save_upload()`.

file_validate_is_image(&\$file)

Эта функция принимает объект файла и предпринимает попытку передачи `$file->filepath` функции `image_get_info()`. Функция возвращает пустой массив, если `image_get_info()` удалось извлечь информацию из файла, или же массив сообщений об ошибках типа `Only JPEG, PNG and GIF images are allowed` (Разрешены только изображения JPEG, PNG и GIF) в случае неудачи. Эта функция может также служить средством проверки достоверности для `file_save_upload()`.

***file_validate_image_resolution(\$file,
\$maximum_dimensions = 0, \$minimum_dimensions = 0)***

Эта функция принимает объект файла и использует `$file->filepath` в нескольких операциях. Если файл является изображением, то функция проверит, не превышает ли оно предельного размера `$maximum_dimensions` и при необходимости попытается изменить размер, если это возможно. Если все выполняется успешно, функция возвратит пустой массив, а объект `$file`, переданный посредством ссылки, будет иметь значение `$file->filesize`, установленное равным новому размеру, если размер изображения был изменен. В противном случае массив будет содержать сообщение об ошибке наподобие `The image is too small; the minimum dimensions are 320x240 pixels` (Изображение слишком мало: минимальный размер 320×240 пикселей). Параметры `$maximum_dimensions` и `$minimum_dimensions` — строки, представляющие ширину и высоту в пикселях, разделенные строчной буквой `x` (например, `640x480` или `85x85`). Значение по умолчанию, равное `0`, указывает отсутствие ограничений на размер. Эта функция может также служить средством проверки достоверности для `file_save_upload()`.

file_validate_name_length(\$file)

Параметр `$file` — это объект файла. Функция возвращает пустой массив, если `$file->filename` не превышает 255 символов. В противном случае она возвращает массив, содержащий сообщение об ошибке с указанием пользователю использовать более короткое имя. Эта функция может также служить средством проверки достоверности для `file_save_upload()`.

file_validate_size(\$file, \$file_limit = 0, \$user_limit = 0)

Эта функция проверяет, что размер файла не превышает максимального предела для файла или совокупного предела для пользователя. Параметр `$file` представляет собой объект файла, который должен содержать значение `$file->filesize`, представляющее собой размер файла в байтах. Параметр `$file_limit` — целое число, представляющее максимальный размер в байтах. Параметр `$user_limit` — целое число, представляющее максимальное совокупное количество байтов, которое разрешено использовать текущему пользователю. `0` означает отсутствие ограничений. При успешном выполнении проверки допустимости будет возвращен пустой массив; в противном случае будет возвращен массив, содержащий сообщение об ошибке. Эта функция может также служить средством проверки достоверности для `file_save_upload()`.

Получение URL для файла

Если имя файла, который был выгружен, известно, и требуется сообщить клиенту URL этого файла, можно воспользоваться следующей функцией.

file_create_url(\$uri)

Эта функция возвратит корректный URL-адрес файла, независимо от того, запущен Drupal в режиме общедоступной или приватной загрузки. Параметр `$uri` — это путь к файлу (например, `sites/default/files/pictures/picture-1.jpg` или `pictures/picture-1.jpg`). Результирующий URL может выглядеть как `http://example.com/sites/default/files/pictures/picture-1.jpg`. Обратите внимание, что абсолютное имя пути для файла не используется. Это облегчает перемещение сайта Drupal с одного места (или сервера) в другое.

Поиск файлов в каталоге

Система Drupal предоставляет мощную функцию `file_scan_directory()`. Она выполняет в каталоге поиск файлов, соответствующих заданному шаблону.

```
file_scan_directory($dir, $mask,  
$options = array(), $depth = 0)
```

Рассмотрим ее сигнатуру.

- `$dir`. Предназначенный для просмотра базовый каталог или URI, без заключительного символа косой черты.
- `$mask`. Шаблон, применяемый к файлам, содержащимся в каталоге. Он представляет собой регулярное выражение.
- `$options`. Ассоциативный массив дополнительных параметров, содержащий перечисленные ниже элементы.
 - `nomask`. Регулярное выражение, используемое функцией `preg_match()` для определения файлов, которые нужно игнорировать. По умолчанию оно принимает значение `"/(\\.\\?ICVS)$/"`.
 - `callback`. Функция обратного вызова, вызываемая для каждого соответствия.
 - `recurse`. Когда значение этого параметра — `TRUE`, просмотр каталогов будет рекурсивно выполняться для всего дерева, начиная с указанного каталога. Значение по умолчанию равно `TRUE`.
 - `key`. Ключ, который должен использоваться для ассоциативного массива файлов. Возможные значения: `"uri"`, для URI файла; `"filename"` для базового имени файла и `"name"` для имени файла без расширения. Значение по умолчанию — `"uri"`.
 - `min_depth`. Минимальная глубина каталогов, из которых требуется вернуть файл. Значения по умолчанию — `0`.
- `$depth`. Текущая глубина рекурсии. Этот параметр используется только внутренне и не должен передаваться.

Возвращаемое значение представляет собой ассоциативный массив объектов. Ключ массива зависит от того, что передается в параметре `key`, и по умолчанию им является `filename`. Ниже приведены некоторые примеры. Поиск любых файлов с окончанием `.css` в каталоге `themes/bluemarine`:

```
$found = file_scan_directory('themes/seven', '$css$');
```

Результирующий массив объектов показан на рис. 14.7.

Изменение значения параметра ключа на имя файла изменяет ключи результирующего массива, как показано в следующем коде и на рис. 14.8.

```
$options = array ('key' => 'filename');  
$found = file_scan_directory('themes/seven', '$css$', $options);
```

Поиск временного каталога

Предпочтительный подход к использованию временного каталога предусматривает применение схемы `temporary://`. Эта схема всегда будет указывать на временный каталог, который был установлен в системе во время процесса инсталляции.

```

Array
(
    [themes/seven/ie.css] => stdClass Object
        (
            [uri] => themes/seven/ie.css
            [filename] => ie.css
            [name] => ie
        )

    [themes/seven/ie6.css] => stdClass Object
        (
            [uri] => themes/seven/ie6.css
            [filename] => ie6.css
            [name] => ie6
        )

    [themes/seven/reset.css] => stdClass Object
        (
            [uri] => themes/seven/reset.css
            [filename] => reset.css
            [name] => reset
        )

    [themes/seven/style.css] => stdClass Object
        (
            [uri] => themes/seven/style.css
            [filename] => style.css
            [name] => style
        )

    [themes/seven/vertical-tabs.css] => stdClass Object
        (
            [uri] => themes/seven/vertical-tabs.css
            [filename] => vertical-tabs.css
            [name] => vertical-tabs
        )
)

```

Рис. 14.7. Результатом вызова `file_scan_directory()` по умолчанию является массив объектов с ключом — полным именем файла

Нейтрализация опасных файлов

Предположим, что вы используете общедоступный метод загрузки файлов, и выгрузка файлов разрешена. Что произойдет, если некто выгрузит файл `bad_exploit.php`? Будет ли он запущен, когда злоумышленник щелкнет на `http://example.com/sites/default/files/bad_exploit.php`? Можно надеяться, что нет, и на то есть три причины. Первая — расширение `.php` никогда не должно присутствовать в списке допустимых расширений для выгружаемых файлов. Вторая — файл `.htaccess`, который должен располагаться в `sites/default/files/.htaccess` (см. главу 17). Однако в некоторых распространенных конфигурациях Apache выгрузка файла `exploit.php.txt` может привести к выполнению файла как кода PHP (см. `http://drupal.org/files/sa-2006-007/advisory.txt`). Это приводит к третьей мере предосторожности: изменению имени файла, чтобы обезопасить его. В качестве защиты от загружаемых на сайт исполняемых файлов применяется описанная ниже функция.

```

Array
(
    [ie.css] => stdClass Object
        (
            [uri] => themes/seven/ie.css
            [filename] => ie.css
            [name] => ie
        )

    [ie6.css] => stdClass Object
        (
            [uri] => themes/seven/ie6.css
            [filename] => ie6.css
            [name] => ie6
        )

    [reset.css] => stdClass Object
        (
            [uri] => themes/seven/reset.css
            [filename] => reset.css
            [name] => reset
        )

    [style.css] => stdClass Object
        (
            [uri] => themes/seven/style.css
            [filename] => style.css
            [name] => style
        )

    [vertical-tabs.css] => stdClass Object
        (
            [uri] => themes/seven/vertical-tabs.css
            [filename] => vertical-tabs.css
            [name] => vertical-tabs
        )
)

```

Рис. 14.8. Теперь результат снабжен ключом по имени файла без полного пути

file_munge_filename(\$filename, \$extensions, \$alerts = TRUE)

Параметр `$filename` — это имя файла, которое должно быть модифицировано. Параметр `$extensions` — разделенная пробелами строка с расширениями файлов. Параметр `$alerts` — булевское значение, которое по умолчанию равно `TRUE` и приводит к выводу посредством функции `drupal_set_message()` предупреждения пользователю о том, что имя файла было изменено. В результате возвращается имя файла со вставленными знаками подчеркивания для исключения возможности запуска.

```

$extensions = variable_get('upload_extensions_default', 'jpg jpeg gif png txt
doc xls pdf ppt pps odt ods odp');
$filename = file_munge_filename($filename, $extensions, FALSE);

```

Результат выполнения:

```
$filename is now exploit.php_.txt
```

Изменение имени файла можно предотвратить, установив значение Drupal-переменной `allow_unsecure_uploads` в файле `settings.php` в 1. Но по соображениям безопасности лучше этого не делать.

`file_unmunge_filename($filename)`

Эта функция пытается отменить действие `file_munge_filename()`, заменяя символ подчеркивания, за которым следует точка, точкой:

```
$original = file_unmunge_filename('exploit.php.txt');
```

Результат выполнения:

```
$original is now exploit.php.txt
```

Обратите внимание, что это приведет также к замене всех специально вставленных сочетаний символов `_.` в исходном имени файла.

Проверка дискового пространства

Следующая функция сообщает о пространстве, занятом файлами.

```
file_space_used($uid=NULL, $status=FILE_STATUS_PERMANENT)
```

Эта функция возвращает общий объем дискового пространства, занятого файлами. Она не выполняет реальную проверку файловой системы, а вместо этого сообщает сумму значений полей `filesize` таблицы `files` в базе данных. Если этой функции передается идентификатор пользователя, запрос ограничивается файлами с соответствующим идентификатором пользователя в таблице `files`. Модуль `upload` помещает эту функцию внутрь функции `upload_space_used()`. Функцию `file_space_used()` нужно вызывать непосредственно, поскольку функция `upload_space_used()` доступна только при активированном модуле `upload`.

Хуки аутентификации для загрузки

Разработчики модулей могут реализовать хук `hook_file_download()` для установки прав доступа на загрузку приватных файлов. Этот хук используется для определения условий отправки файла браузеру и возвращает дополнительные заголовки для Drupal, добавляемые в ответ на HTTP-запрос файла. Краткий обзор процесса загрузки с применением реализации `hook_file_download()` в модуле `user` показан на рис. 14.9.

Поскольку Drupal вызывает все модули, содержащие функцию `hook_file_download()`, при каждой загрузке, важно определить область действия хука. Например, рассмотрим функцию `user_file_download()`, которая отвечает за загрузку файла только в случае, если файл, подлежащий загрузке, находится в каталоге `pictures`. Если это так, она добавляет заголовки к запросу.

```
function user_file_download($uri) {
  if (strpos(file_uri_target($uri),
    variable_get('user_picture_path', 'pictures') . '/picture-') === 0) {
    $info = image_get_info($uri);
    return array('Content-Type' => $info['mime_type']);
  } else {
    return -1;
  }
}
```

Реализации `hook_file_download()` должны возвращать заголовки, если запрос разрешен, или значение -1, если доступ к файлу запрещен. Если ни один модуль не отвечает на хук, Drupal возвращает ошибку 404 Not Found (Не найдено).

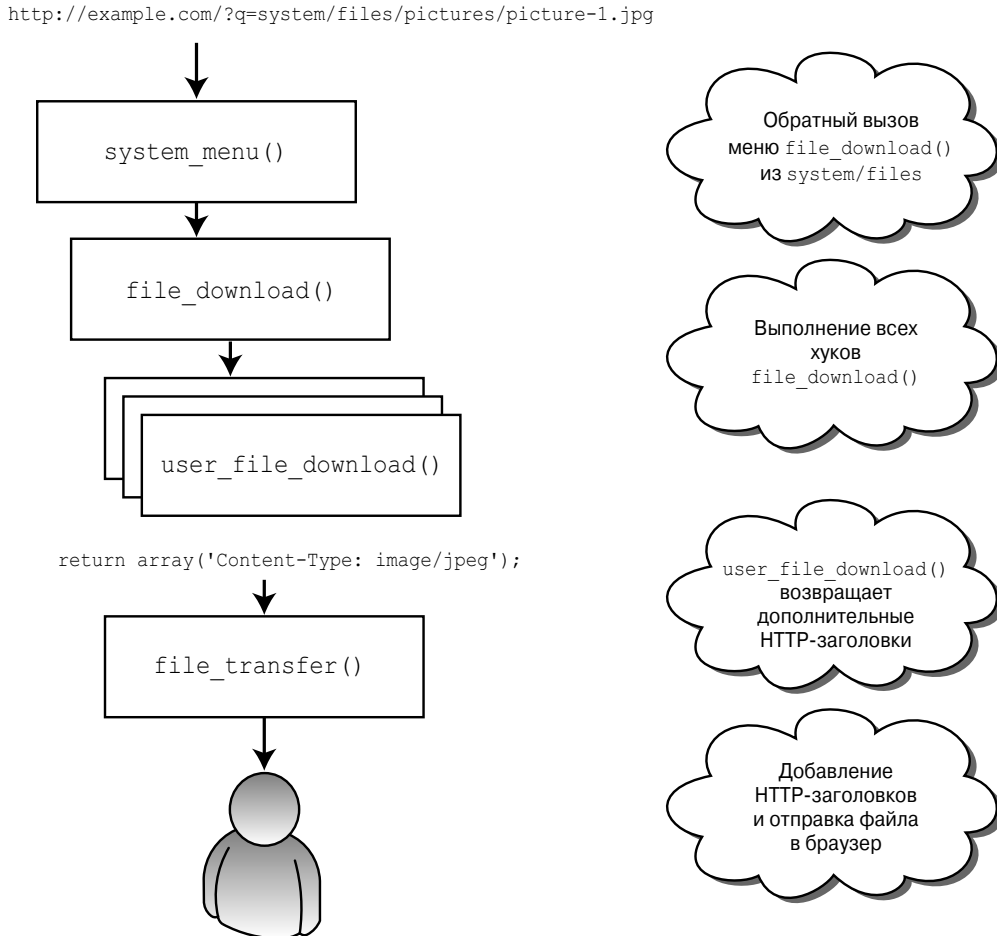


Рис. 14.9. Жизненный цикл запроса на загрузку приватного файла

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Понимать, в чем состоит различие между общедоступными и приватными файлами.
- Знать, какие дополнительные модули используются для обработки изображений, видео- и аудио-контента.
- Знать, что собой представляет схема базы данных для хранения файлов.
- Знать общие функции манипулирования файлами.
- Понимать, какие хуки аутентификации используются при загрузке приватных файлов.

ГЛАВА 15

Работа с таксономией

Таксономия (taxonomy) — это практические приемы и наука классификации. Зачастую этот аспект остается непонятым и не используется в должной мере — т.е. множество людей оставляет без внимания мощное средство, которое могло бы облегчить их участь как разработчиков и администраторов сайтов. Несколько быстрых примеров могут облегчить понимание того, что собой представляет таксономия и зачем она может потребоваться.

Одно из наиболее распространенных применений таксономии — пометка (тегирование) контента определенными словами или выражениями. Например, часть контента можно пометить тегом “замечательно подходит для подростков”. Чтобы найти весь контент, который подходил для подростков, было бы достаточно щелкнуть на этом теге. Среда Drupal найдет весь контент, помеченный упомянутым тегом, и визуализирует его в виде списка. Еще одним способом применения таксономии является облегчение создателям контента на веб-сайте задачи определения места размещения этого контента на веб-сайте посредством выбора из списка заранее определенных терминов. Например, может быть определен таксономический термин “объявления на домашней странице”. Если автор контента назначает этот термин создаваемому им элементу контента, представление, отображающее объявления на домашней странице, визуализирует данную статью без каких-либо действий со стороны автора, помимо выбора таксономического термина.

Хотя Drupal предоставляет множество готовых таксономических решений, в ряде случаев может требоваться использовать функции таксономии из внешнего модуля или расширить функциональные возможности, предоставляемые модулем Taxonomy. В этой главе будет показано, как использовать API-интерфейсы для работы с таксономией, расширяя таксономические функции в рамках модуля.

Структура таксономии

Таксономия состоит из двух основных элементов — словарей и терминов. Словари удобно представлять в виде контейнеров для наборов связанных слов или словосочетаний. Например, можно было бы создать словарь “профессиональный спорт”, а внутри него такие слова или выражения, как “футбол”, “баскетбол”, “хоккей”, “крикет”, “регби” и “гольф”. Каждое слово или выражение является термином. После создания словаря и связанных терминов можно создать соответствующее поле в требуемом типе (типах) контента.

Создание словаря

Чтобы создать словарь, перейдите к странице Structure⇒Taxonomy (Структура⇒Таксономия) и щелкните на ссылке Add vocabulary (Добавить словарь). Форма для создания словаря состоит из полей Name (Имя) и Description (Описание). Просто введите имя словаря и необязательное описание, а затем щелкните на кнопке Save (Сохранить).

Создание терминов

Чтобы создать термины, перейдите к странице Structure⇒Taxonomy и щелкните на ссылке add terms (добавить термины) для выбранного словаря. На странице создания термина введите в поле Name of the term (Имя термина) имя, например, баскетбол, и дополнительно описание и псевдоним URL-адреса. По умолчанию URL, используемый для доступа ко всему контенту, связанному с таксономическим термином, структурируется как taxonomy/term/9. Для создания URL, более удобного для пользователя, такого как professionalsports/basketball, можно применять поле псевдонима URL.

Назначение словаря типу контента

Таксономические словари назначаются типам контента, что позволяет авторам выбирать или вводить термины для ассоциирования с создаваемым ими фрагментом контента. Таксономические словари присваиваются типам контента в виде поля. Чтобы связать словарь с типом контента, перейдите к странице Structure⇒Content types и щелкните на ссылке manage fields (управление полями) для того типа контента, к которому требуется добавить словарь. Интерфейс для создания нового поля типа Term reference (Ссылка на термин), связанного с таксономией, показан на рис. 15.1.

The screenshot shows the 'Add new field' form in Drupal. It includes the following elements:

- A plus icon and the text 'Add new field'.
- A text input field with the value 'Sport being played' and a label 'Label' below it.
- A text input field with the value 'sport' and a label 'Field name (a-z, 0-9, _)' below it.
- A dropdown menu with 'Term reference' selected and a label 'Type of data to store.' below it.
- A dropdown menu with 'Select list' selected and a label 'Form element to edit the data.' below it.

Рис. 15.1. Форма добавления словаря к типу контента

Следующий шаг процесса — выбор таксономического словаря, с которым ассоциируется поле. На следующем экране в процессе представлен простой список выборки, в котором можно выбрать словарь для использования. Выберите словарь и щелкните на кнопке Save field settings (Сохранить настройки поля).

Заключительный шаг в процессе создания основанного на таксономии поля — определение того, обязательно ли поле. То есть автор должен выбрать один или более терминов, справочный текст, отображаемый под списком терминов, значение по умолчанию, которое должно быть выбрано при отображении формы создания контента, и количество значений, которые могут быть выбраны автором. Параметры, которые могут быть установлены на этом последнем экране в процессе создания таксономического поля, показаны на рис. 15.2.

После сохранения поля форма создания контента будет содержать новое поле, в котором автор может произвести выбор из списка терминов, связанных со словарем.

EVENT SETTINGS
 These settings apply only to the *Sport being played* field when used in the *Event* type.

Label *

☐ Required field

Help text

Instructions to present to the user below this field on the editing form.
 Allowed HTML tags: <a> <big> <code> <del> <i> <ins> <pre> <q> <small> <sub> <sup> <tt> <p>

DEFAULT VALUE
 The default value for this field, used when creating new content.
 Sport being played

SPORT BEING PLAYED FIELD SETTINGS
 These settings apply to the *Sport being played* field everywhere it is used.

 Number of values

Maximum number of values users can enter for this field.

 Vocabulary *

The vocabulary which supplies the options for this field.

Рис. 15.2. Параметры конфигурации для нового таксономического поля

Виды таксономии

Существует несколько видов таксономии. Самый простой из них — список терминов, а самый сложный включает множественные иерархические отношения. Кроме того, термины могут быть синонимами или быть связанными с другими терминами. Начнем с самой простой разновидности.

Одноуровневый словарь

Словарь, который состоит только из списка терминов, очень прост. В табл. 15.1 показано, как в простом одноуровневом словаре, который мы назовем *Programming Languages* (Языки программирования), можно классифицировать некоторые языки программирования.

Иерархический словарь

Теперь рассмотрим концепцию *иерархии*, согласно которой каждый термин может быть связан определенным отношением с другим термином (табл. 15.2).

Таблица 15.1. Простые термины в словаре

Идентификатор термина	Имя термина
1	C
2	C++
3	Cobol

Таблица 15.2. Иерархические термины в словаре (дочерние термины размещены под своими родительскими терминами)

Идентификатор термина	Имя термина
1	Object-Oriented
2	C++
3	Smalltalk
4	Procedural
5	C
6	Cobol

На рис. 15.3 показаны иерархические отношения в явном виде. В этом примере Procedural — родительский, а Cobol — дочерний термин. Обратите внимание, что каждый термин имеет собственный идентификатор, независимо от того, является он родительским или дочерним.



Рис. 15.3. Термины в иерархическом словаре связаны между собой отношениями “родительский–дочерний”

Организовать термины в иерархии при создании термина можно выбором родительского термина из поля Parents (Родительские) в разделе Advanced options (Дополнительные параметры) формы Add term (Добавить термин) или перетаскиванием термина в требуемую позицию. После добавления более чем одного термина интерфейс перетаскивания становится доступным на странице Administer⇒Content management⇒Taxonomy (Администрирование⇒Управление контентом⇒Таксономия) при щелчке на ссылке list terms (список терминов) для словаря, с которым выполняется работа. Интерфейс перетаскивания показан на рис. 15.4.

Множественная иерархия

Словарь может иметь не одну, а несколько иерархий. Это означает всего лишь, что термин может иметь более чем одного родителя. Например, предположим, что в словарь языков программирования вы добавляете язык PHP.

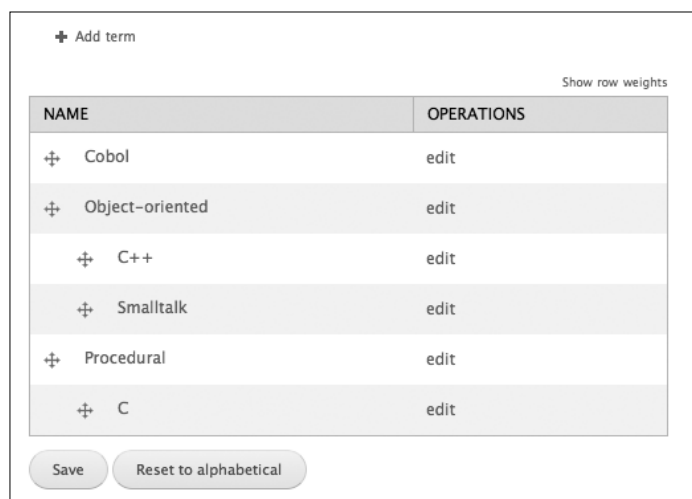


Рис. 15.4. Термины могут быть упорядочены в иерархии с помощью интерфейса перетаскивания

Изначально этот язык был процедурным, но в его новых версиях появились объектно-ориентированные возможности. Следует его классифицировать как объектно-ориентированный или как процедурный? Благодаря множественным иерархическим отношениям, можно сделать и то, и другое, как показано на рис. 15.5.

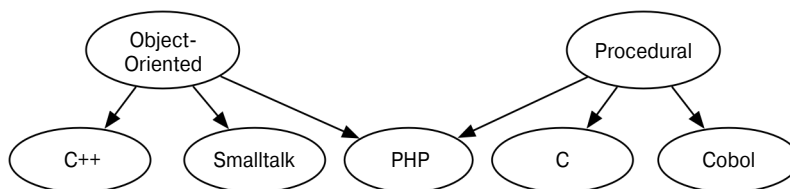


Рис. 15.5. Во множественном иерархическом словаре термины могут иметь более одного родителя

На стадии планирования веб-сайта следует уделить должное время продумыванию вариантов использования таксономии, чтобы определить, какой вид словаря необходим.

Поскольку отображение множественного иерархического словаря в интерфейсе пользователя затруднено, при выборе нескольких родителей для термина Drupal предупреждает, что интерфейс перетаскивания (см. рис. 15.4) будет отключен. Это предупреждение показано на рис. 15.6.

Просмотр контента по термину

Ноды, ассоциированные с определенным термином, всегда можно просмотреть, обратившись к URL термина, если только модуль не переопределит это представление. Например, в адресе <http://example.com/?q=taxonomy/term/5> значение 5 — это идентификатор термина, который требуется просмотреть. Результатом будет список, содержащий заголовки и тизеры каждой ноды, маркированной этим термином.

Set multiple term parents? ✕

LIST **EDIT** **MANAGE FIELDS** **MANAGE DISPLAY**

Adding multiple parents to a term will cause the *Programming Languages* vocabulary to look for multiple parents on every term. Because multiple parents are not supported when using the drag and drop outline interface, drag and drop will be disabled if you enable this option. If you choose to have multiple parents, you will only be able to set parents by using the term edit form.

You may re-enable the drag and drop interface at any time by reducing multiple parents to a single parent for the terms in this vocabulary.

URL alias

Optionally specify an alternative URL by which this term can be accessed. Use a relative path and don't add a trailing slash or the URL alias won't work.

Set multiple parents Cancel

Рис. 15.6. Выбор нескольких родителей для термина приводит к отключению интерфейса перетаскивания

Использование операций AND и OR в URL

Синтаксис создания URL-адресов таксономии поддерживает операции AND и OR, представляемые, соответственно, символами запятой (,) и плюса (+). Ниже приведен ряд примеров.

Чтобы показать все ноды, которым были назначены идентификаторы терминов 5 и 6, применяется следующий URL:

```
http://example.com/?q=taxonomy/term/5,6
```

Чтобы отобразить все ноды, которым были назначены идентификаторы терминов 1, 2 или 3, нужно использовать такой URL:

```
http://example.com/?q=taxonomy/term/1+2+3
```

Совет. Используйте модуль `path`, чтобы установить дружественные псевдонимы URL для применяемых URL таксономии. Тогда они не будут завершаться столь раздражающими цифрами.

Определение глубины для иерархических словарей

В предыдущих примерах применялся один неявный параметр. Например, следующий URL:

```
http://example.com/?q=taxonomy/term/5
```

в действительности означает:

```
http://example.com/?q=taxonomy/term/5/0
```

Здесь завершающий 0 является количеством просматриваемых уровней иерархии при подготовке результирующего набора для отображения; аргумент `all` означал бы, что в процесс поиска должны были бы быть включены все уровни. Предположим, что имеется иерархический словарь, описанный в табл. 15.3.

Таблица 15.3. Географический иерархический словарь (дочерние термины размещены под своими родительскими с отступом)

Идентификатор термина	Имя
1	Canada (Канада)
2	British Columbia (Британская Колумбия)
3	Vancouver (Ванкувер)
4	Ontario (Онтарио)
5	Toronto (Торонто)

Первый уровень иерархии — это страна (Канада); он имеет два дочерних термина: провинции Британская Колумбия и Онтарио. Каждая провинция имеет один дочерний термин — канадские города, в которых Drupal развивается наиболее быстрыми темпами. Изменение параметра глубины URL ведет к следующему эффекту. Все ноды, маркированные термином Vancouver, будут совместно использовать следующие URL:

```
http://example.com?q=taxonomy/term/3 или
http://example.com?q=taxonomy/term/3/0
```

Чтобы отобразить все ноды, маркированные термином British Columbia (но не Vancouver), нужно использовать такой URL:

```
http://example.com?q=taxonomy/term/2
```

Следующий URL применим ко всем нодам, которые маркированы названием провинции British Columbia (Британская Колумбия) и любым городом Британской Колумбии (обратите внимание, что глубина уровня иерархии установлена равной 1):

```
http://example.com?q=taxonomy/term/2/1
```

На заметку! Результирующий набор отображается в виде обычного списка нод. Если требуется, чтобы заголовки и/или тизеры нод отображались *иерархически*, понадобится создать специальную функцию темы или воспользоваться модулем views (<http://drupal.org/project/views>).

Автоматические RSS-каналы

Каждый термин имеет автоматически создаваемый RSS-канал, который отображает последние ноды, маркированные данным термином. Например, канал для термина с идентификатором 3 можно найти по следующему адресу:

```
http://example.com/?q=taxonomy/term/3/feed
```

Хранение таксономий

Если требуются больше возможностей, чем предоставляется встроенными таксономиями, придется разобраться в том, как таксономии хранятся в базе данных. В типичной базе данных, отличной от Drupal, одноуровневую таксономию можно создать, просто добавив столбец в таблицу базы данных. Как уже упоминалось, Drupal добавляет таксономию посредством нормализованных таблиц базы данных. Структура таблиц показана на рис. 15.7.

taxonomy_vocabulary vid name machine_name description hierarchy module weight	taxonomy_index nid tid sticky created
taxonomy_term_data tid vid name description format weight	taxonomy_term_hierarchy tid parent
field_data_field_xxxxxx etid bundle deleted entity_id revision_id language delta field_related_term_tid	field_revision_field_xxxxxx etid bundle deleted entity_id revision_id language delta field_xxxxxx_tid

Рис. 15.7. Таблицы таксономии Drupal. Столбец `vid` в таблице `taxonomy_term_data` представляет идентификатор версии из таблицы `node_revisions`, а не на идентификатор словаря

Система хранения таксономии Drupal состоит из следующих таблиц.

- *taxonomy_vocabulary*. Эта таблица хранит информацию о словаре, который можно редактировать через интерфейс Таксоному (Таксономия) системы Drupal.
- *taxonomy_index*. Эта таблица хранит отношения между нодами и таксономическими терминами.
- *taxonomy_term_data*. Эта таблица содержит фактическое имя термина, название словаря, в котором он находится, необязательное описание и вес, определяющий его позицию в списках терминов, предоставляемых пользователю для выбора термина (например, на форме отправки ноды).
- *taxonomy_term_hierarchy*. Эта таблица содержит идентификатор термина, а также идентификатор его родительского термина. Если термин находится в корне (т.е. не имеет родителя), идентификатор родителя равен 0.
- *field_data_field_xxxxxx*. Эта таблица используется для хранения информации, связанной с термином, где *xxxxxx* — имя поля, определенного для конкретного словаря.

Словари на основе модулей

В дополнение к словарям, которые могут создаваться с помощью команды `Admin⇒Structure⇒Taxonomy⇒Add` (Администрирование⇒Структура⇒Таксономия⇒Добавить), модули могут использовать таблицы таксономии для хранения своих собственных словарей. Например, модуль `Forum` (Форум) использует таблицы таксономии для хранения словаря контейнеров и форумов.

Модуль, владеющий словарем, указывается в столбце `module` таблицы `taxonomy_vocabulary`. Обычно этот столбец будет содержать элемент `taxonomy`, поскольку именно модуль таксономии управляет большинством словарей.

Создание словаря на основе модуля

Давайте рассмотрим пример словаря на основе модуля. Модуль `Forum` (Форум) использует таксономию в качестве механизма организации тем форумов. Он создает свой словарь программно (см. `modules/forum/forum.install`), как показано в следующем примере, и становится владельцем словаря, устанавливая имя этого модуля (без окончания `.module`) в качестве значения ключа `module` массива `$vocabulary`.

```
// Создать словарь форума, если он не существует.
$vocabulary = taxonomy_vocabulary_load(variable_get('forum_nav_vocabulary', 0));
if (!$vocabulary) {
  $edit = array(
    'name' => t('Forums'),
    'machine_name' => 'forums',
    'description' => t('Forum navigation vocabulary'),
    'hierarchy' => 1,
    'module' => 'forum',
    'weight' => -10,
  );
  $vocabulary = (object) $edit;
  taxonomy_vocabulary_save($vocabulary);
  variable_set('forum_nav_vocabulary', $vocabulary->vid);
}
```

Уведомление об изменениях в словаре с помощью хуков таксономии

Если вы предусмотрели словарь для своего модуля, понадобится получать информацию относительно любых изменений, вносимых посредством стандартного пользовательского интерфейса `Taxonomy`. Может также требоваться получать информацию о внесении изменений в существующий словарь, поддерживаемый модулем `taxonomy.module`.

Во всех случаях узнавать об изменениях в словарях можно, реализовав хук `hook_taxonomy()`. Следующий модуль содержит реализацию `hook_taxonomy()`, которая будет сообщать об изменениях в словаре по электронной почте. Файл `taxonomymonitor.info` имеет следующий вид:

```
name = Taxonomy Monitor
description = Sends email to notify of changes to taxonomy vocabularies.
package = Pro Drupal Development
dependencies[] = taxonomy
core = 7.x
files[] = taxonomymonitor.module
```

А вот так выглядит модуль `taxonomymonitor.module`:

```
<?php

/**
 * @file
 * Модуль, который уведомляет по электронной почте об изменениях в таксономии.
 */

/**
 * Реализация хука hook_term_insert().
 */
function taxonomymonitor_term_insert($term) {
  _send_notification('term', 'added', $term->name);
}

/**
 * Реализация хука hook_term_update().
 */
function taxonomymonitor_term_update($term) {
  _send_notification('term', 'updated', $term->name);
}

/**
 * Реализация хука hook_term_delete().
 */
function taxonomymonitor_term_delete($term) {
  _send_notification('term', 'deleted', $term->name);
}

/**
 * Реализация хука hook_vocabulary_insert().
 */
function taxonomymonitor_vocabulary_insert($vocabulary) {
  _send_notification('vocabulary', 'added', $vocabulary->name);
}

/**
 * Реализация хука hook_vocabulary_update().
 */
function taxonomymonitor_vocabulary_update($vocabulary) {
  _send_notification('vocabulary', 'updated', $vocabulary->name);
}

/**
 * Реализация хука hook_vocabulary_delete().
 */
function taxonomymonitor_vocabulary_delete($vocabulary) {
  _send_notification('vocabulary', 'deleted', $vocabulary->name);
}

/**
 * Регистрация изменений в таксономии с помощью watchdog и отправка
 * по электронной почте уведомления с описанием изменения.
 */
function _send_notification($type, $action, $name) {
  watchdog("Taxonomy Monitor", t("A $type named $name was $action"),
```

```

$variables=array(),
$severity=WATCHDOGNOTICE, $link=NULL);
  $to = variable_get('site_mail','');
  $subject = t("There was a change to taxonomy");
  $body = t("A $type named $name was $action");
  drupal_mail('taxonomymonitor', $to, $subject, $body);
}

```

Дополнительно можно было бы уведомлять и об имени пользователя, внесшего изменения.

Общие задачи

Ниже описаны некоторые общие задачи, которые могут встретиться при работе с таксономией.

Отображение терминов таксономии, связанных с нодой

В Drupal 7 термины таксономии представляют собой всего лишь поля, связанные с нодой. Чтобы выяснить значение таксономических терминов, вначале нужно получить кое-какую информацию о типе контента, связанного с просматриваемой нодой — в частности, имя поля, используемого для хранения терминов таксономии. В качестве примера применяется ранее созданная нода *Article* (Статья), которой назначены теги. После создания ноды и получения идентификатора новой ноды (в примере он равен 2) термины таксономии, связанные с этой нодой, можно было бы вывести на экран, используя следующий фрагмент кода. Фрагмент вставляется в новый блок, в качестве входного формата устанавливается PHP-код, после чего блок назначается области на странице.

```

<?php
/**
 * Отображение терминов таксономии, связанных с заданной нодой.
 */
$nid = 2;
$node = node_load($nid);
$result = field_view_field('node', $node, 'field_tags', $mode = array('default'));
print render($result);

```

В приведенном коде вызывается функция `field_view_field()` с передачей ей типа объекта, который содержит поле (ноду), объект, содержащий поле (загруженный объект ноды (`$node`)), имя поля (`field_geographic_location`) и режим отображения (`default` — режим по умолчанию).

Построение собственных запросов таксономии

Если требуется сгенерировать тот или иной список нод, может быть желательным несколько упростить ситуацию. Например, если бы система Drupal хранила термины таксономии в таблице `node`, было бы достаточно использовать следующий запрос:

```
SELECT * FROM node WHERE vocabulary = 1 and term = 'cheeseburger'
```

Для обеспечения подобной гибкости разработчику Drupal пришлось бы затратить несколько больше усилий. Поэтому вместо того, чтобы выдавать простые запросы, подобные приведенному, следует научиться выполнять запросы таблиц таксономии, используя операции `JOIN`.

Использование функции `taxonomy_select_nodes()`

Прежде чем приступить к написанию запроса, подумайте, нельзя ли получить требуемый результат с помощью существующих функций. Например, чтобы получить заголовки нод, помеченных терминами с идентификаторами 5 и 6, можно воспользоваться функцией `taxonomy_select_nodes()`:

```
$result = taxonomy_select_nodes($tids);
$titles = array();
foreach($result as $nid) {
    $node = node_load($nid);
    $titles[] = $node->title;
}
```

Функции таксономии

В следующих разделах рассмотрены функции, которые могут оказаться полезными для вашего модуля.

Извлечение информации о словарях

Описанные в следующих разделах встроенные функции извлекают информацию о словарях в виде объектов данных словарей или в виде массива таких объектов.

taxonomy_vocabulary_load(\$vid)

Эта функция извлекает один словарь (параметр `$vid` — идентификатор словаря) и возвращает объект словаря. Она выполняет также внутреннее кэширование объектов словарей, поэтому многократные запросы к одному и тому же словарю оказываются сравнительно недорогими в плане ресурсов. Эта функция также является специальной функцией загрузки с точки зрения системы меню Drupal (подробнее система меню описана в главе 4). Поскольку словари таксономии являются объектами, функция `taxonomy_vocabulary_load` представляет собой лишь оболочку функции `entity_load`.

taxonomy_get_vocabularies()

Функция `taxonomy_get_vocabularies($type)` извлекает массив всех объектов словарей.

Добавление, изменение и удаление словарей

Следующие функции создают, изменяют и удаляют словари. Они возвращают код состояния, который является одной из констант Drupal: `SAVED_UPDATED` или `SAVED_NEW`.

taxonomy_vocabulary_save(\$vocabulary)

Эта функция создает новый словарь или обновляет существующий. Параметр `$vocabulary` — объект словаря, содержащий перечисленные ниже ключи.

- *vid*. Идентификатор словаря.
- *name*. Имя словаря.
- *machine_name*. Внутреннее имя этого словаря в системе Drupal.
- *description*. Описание словаря.
- *hierarchy*. Устанавливается в 0 при отсутствии иерархии, в 1 — для одной иерархии и в 2 — для множества иерархий.

- *module*. Имя модуля, ответственного за данный словарь; если этот ключ не передан, по умолчанию он примет значение *taxonomy*.
- *weight*. Вес словаря; влияет на его позицию внутри набора полей *Vocabularies* (Словари) в форме отправки ноды.
- *rdf_mapping*. Массив, определяющий способ отображения терминов.

Функция `taxonomy_save_vocabulary($vocabulary)` возвращает константу `SAVED_NEW` или `SAVED_UPDATED`.

taxonomy_vocabulary_delete(\$vid)

Параметр *\$vid* этой функции — это идентификатор словаря. Удаление словаря ведет к удалению всех его терминов за счет вызова функции `taxonomy_del_term()` для каждого термина. Функция `taxonomy_vocabulary_delete($vid)` возвращает константу `SAVED_DELETED`.

Извлечение информации о терминах

Встроенные функции, описанные в следующих разделах, извлекают информацию о терминах, обычно в виде объектов или массивов объектов.

taxonomy_load_term(\$tid)

Эта функция извлекает термин (параметр *\$tid* — идентификатор термина) и возвращает объект термина. Она выполняет внутреннее кэширование объектов терминов, поэтому многократные запросы одного и того же термина влекут за собой незначительное снижение производительности. Структура объекта термина выглядит следующим образом:

```
$term = taxonomy_term_load(3);
var_dump($term);
object(stdClass)#65 (9) {
  ["tid"]=> string(1) "3"
  ["vid"]=> string(1) "3"
  ["name"]=> string(16) "British Columbia"
  ["description"]=> string(38) "A western province of stunning beauty."
  ["format"]=> string(1) "3"
  ["weight"]=> string(1) "0"
  ["vocabulary_machine_name"]=> string(20) "geographic_locations"
  ["field_related_term"]=> array(1) {
    ["und"]=> array(1) {
      [0]=> array(1) {
        ["tid"]=> string(1) "2"
      }
    }
  }
  ["rdf_mapping"]=> array(5) {
    ["rdftype"]=> array(1) {
      [0]=> string(12) "skos:Concept"
    }
    ["name"]=> array(1) {
      ["predicates"]=> array(2) {
        [0]=> string(10) "rdfs:label"
        [1]=> string(14) "skos:prefLabel"
      }
    }
  }
}
```

```

["description"]=> array(1) {
  ["predicates"]=> array(1) {
    [0]=> string(15) "skos:definition"
  }
}
["vid"]=> array(2) {
  ["predicates"]=> array(1) {
    [0]=> string(13) "skos:inScheme"
  }
  ["type"]=> string(3) "rel"
}
["parent"]=> array(2) {
  ["predicates"]=> array(1) {
    [0]=> string(12) "skos:broader"
  }
  ["type"]=> string(3) "rel"
}
}
}

```

taxonomy_get_term_by_name(\$name)

Функция `taxonomy_get_term_by_name($name)` ищет термины, соответствующие некоторой строке (передаваемой в параметре `$name`). Символы пробелов удаляются из `$name`, а соответствия устанавливаются с помощью запроса `WHERE LOWER(t.name) = LOWER($name)`. Эта функция возвращает массив объектов терминов.

Добавление, изменение и удаление терминов

Следующие функции позволяют создавать, изменять и удалять термины. Они возвращают код состояния, который является одной из констант Drupal: `SAVED_UPDATED`, `SAVED_NEW` или `SAVED_DELETED`.

taxonomy_term_save(\$term)

Эта функция создает новый термин или обновляет существующий. `$term` — это объект термина.

- *name*. Имя термина.
- *description*. Описание термина. Это значение не применяется в пользовательском интерфейсе Drupal по умолчанию, но может использоваться вашим модулем или сторонними модулями.
- *vid*. Идентификатор словаря, которому принадлежит этот термин.
- *weight*. Вес данного термина. Он влияет на порядок, в котором термины отображаются в полях выбора терминов.
- *relations*. Необязательный массив идентификаторов терминов, с которыми связан данный термин.
- *parent*. Может быть строкой, представляющей идентификатор родительского термина, или массивом, содержащим либо строки, представляющие идентификаторы родительских терминов, либо подмассив, содержащий строки, которые представляют идентификаторы родительских терминов. Этот ключ является необязательным.
- *vocabulary_machine_name*. Машинное имя словаря, связанного с этим термином.
- *tid*. Идентификатор термина. Если этот ключ не передается, будет создан новый термин. Эта функция возвращает константу `SAVED_NEW` или `SAVED_UPDATED`.

taxonomy_term_delete(\$tid)

Эта функция удаляет термин (параметр *\$tid* — идентификатор данного термина). Если термин находится в иерархическом словаре и имеет дочерние термины, те из них, которые имеют единственного родителя, также будут удалены.

Извлечение информации об иерархии терминов

При работе с иерархическими словарями могут пригодиться функции, описанные в следующих разделах.

taxonomy_get_parents(\$tid, \$key)

Эта функция находит непосредственных предков термина (параметр *\$tid* — идентификатор данного термина). По умолчанию параметр *\$key* принимает значение *tid* и является столбцом таблицы *term_data* (*tid, vid, name, description, weight*). Функция *taxonomy_get_parents(\$tid, \$key)* возвращает ассоциативный массив объектов терминов с ключами *\$key*.

taxonomy_get_parents_all(\$tid)

Эта функция находит всех предков термина (параметр *\$tid* — идентификатор данного термина). Функция возвращает массив объектов терминов.

taxonomy_get_children(\$tid, \$vid, \$key)

Функция *taxonomy_get_children(\$tid, \$vid, \$key)* находит все дочерние термины данного термина (параметр *\$tid* — идентификатор термина). Параметр *\$vid* является необязательным; если в нем передается идентификатор словаря, дочерние термины будут ограничены только этим словарем (обратите внимание, что это важно только для терминов, имеющих несколько родителей в различных словарях, что случается довольно редко). По умолчанию параметр *\$key* принимает значение *tid* и является столбцом таблицы *term_data* (*tid, vid, name, description, weight*). Эта функция возвращает ассоциативный массив объектов терминов с ключами *\$key*.

taxonomy_get_tree(\$vid, \$parent, \$depth, \$max_depth)

Эта функция генерирует иерархическое, древовидное представление словаря. Параметр *\$vid* — идентификатор словаря, для которого генерируется дерево. Можно определить параметр *\$parent*, если нужно строить не все дерево для словаря, а только ту его часть, которая находится под идентификатором термина, заданным в *\$parent*. Параметр *\$depth* предназначен для внутреннего использования и по умолчанию имеет значение *-1*. Параметр *\$max_depth* — целое число, которое указывает количество уровней дерева, подлежащих возврату, и его значение по умолчанию равно *NULL*, что указывает на все уровни. Эта функция возвращает массив объектов терминов с добавленными ключами *depth* и *parent*. Ключ *depth* — целое число, указывающее уровень иерархии, который в дереве занимает термин, а ключ *parent* — массив идентификаторов родительских терминов данного термина. Например, вот как можно получить результаты для словаря с идентификатором 2 из табл. 14.3:

```
$vid = 2;
print_r(taxonomy_get_tree($vid));
Результат будет следующим:
Array (
    [0] => stdClass Object (
        [tid] => 1
        [vid] => 2
```

```

    [name] => Canada
    [description] => A mari usque ad mare.
    [format] => 3
    [weight] => 0
    [depth] => 0
    [parents] => Array (
        [0] => 0 )
    )

    [1] => stdClass Object (
        [tid] => 4
        [vid] => 2
        [name] => Ontario
        [description] => Ut incepit fidelis sic permanet.
        [format] => 3
        [weight] => 0
        [depth] => 1
        [parents] => Array (
            [0] => 1 )
        )

    [2] => stdClass Object (
        [tid] => 5
        [vid] => 2
        [name] => Toronto
        [description] => Diversity Our Strength.
        [weight] => 0
        [depth] => 2
        [parents] => Array (
            [0] => 4 )
        )

    [3] => stdClass Object (
        [tid] => 2
        [vid] => 2
        [name] => British Columbia
        [description] => Splendor sine occasu.
        [format] => 3
        [weight] => 0
        [depth] => 1
        [parents] => Array (
            [0] => 1 )
        )

    [4] => stdClass Object (
        [tid] => 3
        [vid] => 2
        [name] => Vancouver
        [description] => By Land, Sea and Air We Prosper.
        [format] => 3
        [weight] => 0
        [depth] => 2
        [parents] => Array (
            [0] => 2 )
        )
    )
)

```

Поиск нод с определенными терминами

Иногда может потребоваться простой способ выполнения запроса для выяснения того, какие ноды содержат определенные термины, или вывода результата такого запроса. Функция `taxonomy_select_nodes($tids, $pager, $limit, $order)` помогает в достижении этой цели. Эта функция ищет ноды, которые соответствуют определенным условиям, создавая и выполняя запрос к базе данных, основанный на заданных параметрах. Она возвращает идентификатор ресурса, указывающий на результаты запроса. Параметр `$tids` — это массив идентификаторов терминов. Параметр `$limit` задает максимальное количество нод, которые нужно найти. Установка параметра `$limit` в `FALSE` ведет к возврату всех нод (без ограничения). Параметр `$pager` — булевское значение, указывающее, будут ли результирующие ноды использовать средство разбиения на страницы, и по умолчанию этот параметр принимает значение `TRUE`. В случае генерации RSS-канала параметру `$pager` можно присвоить значение `FALSE`. Параметр `$order` содержит ассоциативный массив условий, которые будут применены к SQL-оператору — например, `'t.created' => 'DESC'` сортирует результаты в порядке убывания даты создания.

В случае поиска множества терминов эта функция может породить значительное количество запросов к базе данных.

Дополнительные ресурсы

Многие модули используют таксономию для разнообразных целей — от добавления контроля доступа (http://drupal.org/project/taxonomy_access) до просмотра динамических категорий (http://drupal.org/project/taxonomy_browser) и показа нод, связанных в блок посредством терминов таксономии (<http://drupal.org/project/similarterms>). Дополнительную информацию о таксономии можно найти в руководстве по Drupal по адресу <http://drupal.org/handbook/modules/taxonomy>. Ознакомьтесь также со списком модулей, связанных с таксономией, по адресу <http://drupal.org/project/Modules/category/71>.

Рекомендуется опробовать на практике модуль `views` (<http://drupal.org/project/views>), особенно те его возможности, которые касаются таксономии.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Понимать, что такое таксономия.
- Понимать, что собой представляют термины, словари и их параметры.
- Понимать, в чем заключается разница между одноуровневыми, иерархическими и множественными иерархическими словарями.
- Создавать URL для поиска терминов таксономии с использованием операций `AND` и `OR`.
- Создавать URL для RSS-каналов терминов таксономии и комбинаций терминов.
- Понимать, как хранится таксономия.
- Знать, как использовать словари в собственном модуле.
- Настраивать модуль для получения уведомлений об изменениях в таксономии.

ГЛАВА 16

Кэширование

При создании страниц для динамических веб-сайтов требуются многочисленные запросы к базе данных на предоставление информации о сохраненном контенте, настройках сайта, текущем пользователе и т.д. Сохранение результатов этих дорогостоящих операций для последующего использования — один из самых легких способов ускорения работы медлительного сайта на уровне приложений. Экономия происходит не только за счет обращений к базе данных: также удастся избежать обработки извлеченной информации в PHP. Встроенный в Drupal API-интерфейс кэширования делает это автоматически для большей части данных ядра и предоставляет множество инструментальных средств для разработчиков Drupal, которые хотят использовать этот API-интерфейс для собственных целей. Например, модуль memcache (<http://drupal.org/project/memcache>) — пример кэширования в памяти, который использует API-интерфейс кэширования.

На заметку! В этой главе описано кэширование в рамках приложения Drupal. Другие уровни кэширования, такие как внутреннее кэширование базы данных (например, кэш запросов MySQL), также могут оказывать существенное влияние на производительность. Об этом упоминается в главе 22.

Когда необходимо кэширование

Важно помнить, что кэширование — это всегда компромисс. Кэширование крупных фрагментов данных заметно повышает производительность, но только в случаях, когда такой фрагмент данных требуется второй или третий раз. Вот почему встроенное полнотрафикное кэширование Drupal используется только для анонимных посетителей; зарегистрированные пользователи часто требуют специализированных версий страниц, и в этом случае кэширование было бы намного менее эффективным. Кэширование мелких фрагментов данных (например, списка популярных статей на нынешний момент времени) обеспечивает менее существенный выигрыш в производительности, но все же ускоряет работу вашего сайта.

Кэширование лучше работает с данными, которые изменяются медленно. Оно вполне подходит для списка самых популярных рассказов за неделю. А вот кэширование списка последних пяти комментариев, посланных в загруженный форум, менее полезно, поскольку эта информация устаревает настолько быстро, что всего несколько посетителей смогут воспользоваться кэшированным списком, прежде чем его понадобится обновить. В худшем случае неудачная стратегия кэширования (например, кэширование слишком часто изменяющихся данных) лишь увеличит накладные расходы сайта вместо того, чтобы их сократить.

Как работает кэширование

Часто модули должны выполнять дорогостоящие запросы базы данных либо вызывать удаленные веб-службы. Вместо использования ресурсов для операций подобного рода в каждом таком случае, модули могут хранить кэш своих данных в одной из корзин (bin), зарезервированных для кэширования в базе данных Drupal. Корзины — это специальные таблицы в базе данных. Ниже перечислены стандартные корзины.

- `cache`. Эта корзина предназначена для хранения универсального кэша. Она используется для хранения переменных, реестра тем, локальной даты, списка простых тестов и т.п.
- `cache_block`. В этой корзине хранится содержимое различных блоков.
- `cache_filter`. В этой корзине хранятся отфильтрованные фрагменты содержимого.
- `cache_form`. В этой корзине хранятся многошаговые формы.
- `cache_menu`. В этой корзине хранится структура видимых навигационных меню отдельных страниц.
- `cache_page`. В этой корзине хранятся страницы, сгенерированные для анонимных пользователей. Данная таблица часто очищается — при каждом изменении страницы или, по крайней мере, при каждой отправке ноды и комментария. Эта корзина — единственная, на которую влияют настройки кэша страниц в панели администрирования.
- `cache_path`. В этой корзине хранятся системные пути, у которых имеются псевдонимы.
- `cache_update`. В этой корзине хранятся доступные версии.

Модули могут также создавать собственную таблицу и хранить данные там. Когда эта информация потребуется в следующий раз, ее можно быстро извлечь с помощью единственного запроса. Как будет показано далее в этой главе, механизм кэширования Drupal является подключаемым, так что хотя здесь мы и говорим о таблицах базы данных, в действительности роль этого механизма может играть и другое хранилище, такое как набор двумерных файлов или кэш, расположенный в памяти.

Таблица, назначенная по умолчанию, в которую ваш модуль может записывать кэшируемую информацию, имеет имя `cache`. Использование этой таблицы — лучший выбор, если нужно сохранить лишь несколько строк кэшируемой информации. При определении новой таблицы кэша для модуля она должна быть структурно идентичной назначаемой по умолчанию таблице `cache`, хотя ее имя будет другим. Для сохранения единообразия рекомендуется предварять имя таблицы префиксом `cache_`. Давайте рассмотрим структуру базы данных таблицы `cache` (табл. 16.1).

На заметку! Определяемая новая таблица кэша для вашего модуля должна быть структурно идентичной назначаемой по умолчанию таблице `cache`.

Столбец `cid` хранит первичный идентификатор кэша для быстрого поиска. Примеры идентификаторов кэша, используемых в ядре Drupal — это URL страницы, которая кэшируется (например, `http://example.com/?q=node/1`), строка и имя темы для кэширования реестра тем (например, `theme_registry:garland`) или даже обычные строки (например, содержимое таблицы `variables` кэшируется с первичным идентификатором кэша, установленным в `variables`). Здесь важно отметить, что идентификатор кэша должен быть уникальным для каждого кэшируемого элемента.

Таблица 16.1. Схема таблицы `cache`

Поле	Тип	Допускаются ли значения NULL	Значение по умолчанию
<code>cid</code>	<code>varchar(255)</code>	Нет	–
<code>data</code>	<code>longblob</code>	Да	–
<code>expire</code>	<code>int</code>	Нет	0
<code>created</code>	<code>int</code>	Нет	0
<code>serialized</code>	<code>smallint</code>	Нет	0

Столбец `data` хранит информацию, которую необходимо кэшировать. Сложные типы данных, такие как массивы или объекты, должны быть сериализованы с помощью PHP-функции `serialize()`, чтобы сохранить свою структуру в базе данных (Drupal выполняет это автоматически).

Столбец `expire` принимает одно из трех значений.

- `CACHE_PERMANENT`. Это значение указывает, что элемент не может быть удален до тех пор, пока не будет вызвана функция `cache_clear_all()` с идентификатором кэша постоянного элемента, который нужно удалить.
- `CACHE_TEMPORARY`. Это значение указывает, что элемент может быть удален, когда в следующий раз будет вызвана функция `cache_clear_all()` для “генеральной” очистки; никакое минимальное время не назначается. Элементы, отмеченные как `CACHE_PERMANENT`, из кэша не удаляются.
- *Временная метка Unix*. Указывает, что элемент должен быть сохранен, по крайней мере, до тех пор, пока не истечет определенное время, после чего он будет вести себя как элемент, отмеченный `CACHE_TEMPORARY`, и станет доступным для удаления.

Столбец `created` — временная метка Unix, указывающая дату создания записи в кэше.

Столбец `serialized` указывает на то, находятся ли данные в столбце в сериализованной форме. Значение 0 указывает на несериализованные данные, в то время как 1 — на сериализованные. Если данные сериализуются, и значение столбца `serialized` равно 1, система кэширования десериализует данные перед возвратом вызывающему коду. Система кэширования автоматически сериализует объект и данные массива, а также устанавливает значение столбца `serialized` в 1, когда данные такого типа кэшируются.

Как кэширование используется в ядре Drupal

Drupal поставляется с десятью таблицами кэша по умолчанию: `cache` хранит копию таблицы `variables` и схемы базы данных и реестр тем; `cache_block` хранит кэшированные копии блоков; `cache_bootstrap` хранит информацию, используемую во время процесса начальной загрузки; `cache_field` хранит информацию о полях; `cache_image` хранит информацию об изображениях; `cache_menu` хранит кэшированные копии навигационных меню; в `cache_filter` сохраняются кэшированные копии контента каждой ноды после того, как он был проанализирован системой фильтра; `cache_form` используется API-интерфейсом форм для предотвращения построения формы когда это возможно; `cache_page` хранит кэшированные копии страниц для анонимных пользо-

вателей; `cache_path` хранит кэшированные псевдонимы URL; и `cache_update` хранит информацию, связанную с текущей версией Drupal и модулями, используемыми в данном сайте. Следует отметить, что настройки Page cache (Кэш страниц) и Block cache (Кэш блоков) меню Configuration⇒Performance (Конфигурация⇒Производительность) оказывают влияние только на таблицы кэша страниц и кэша блоков, но не на остальные компоненты кэша Drupal. Другими словами, фильтры, меню и настройки модулей всегда кэшируются.

Система меню

Система меню кэширует информацию маршрутизации, которая связывает пути Drupal с обратными вызовами. Любое меню, созданное модулем меню, кэшируется, независимо от того, включено ли кэширование страниц Drupal. Поэтому, чтобы очистить кэш меню, нужно использовать кнопку Clear cached data (Очистить кэшированные данные) на странице Configuration⇒Performance или вызвать функцию `menu_cache_clear_all()`. Если изменения были внесены в меню, которые будут влиять на блоки, вместо этого может потребоваться вызов более агрессивной функции `menu_rebuild()`; при перестройке меню кэш меню очищается. Примеры меню включают в себя главное (Main) и дополнительное (Secondary) меню Drupal, а также навигационное меню пользователя. Меню кэшируются для каждого пользователя и для каждой локали в отдельности. Система меню подробно рассматривалась в главе 4.

Кэширование фильтрованного текста

При создании или редактировании ноды ее контент подвергается обработке различными фильтрами, связанными с ее входным форматом. Например, формат фильтра HTML преобразует разрывы строк в HTML-теги `<p>` и `<br/>`, а также удаляет вредоносный HTML-код. Выполнение этих действий для каждого отдельного представления ноды было бы слишком дорогостоящей операцией. Поэтому фильтры применяются к ноде сразу после ее создания или редактирования, и этот контент кэшируется в таблице `cache_filter` базы данных, независимо от того, включено ли кэширование страниц Drupal. Входные форматы подробнее описаны в главе 11.

Совет. Кэш фильтра является причиной, по которой изменения устанавливаемой по умолчанию длины тизеров ноды в административном интерфейсе вступают в силу только после повторного сохранения каждой ноды. Быстрый способ решения этой проблемы — опустошение таблицы `cache_filter`, в результате чего контент всех нод будет проанализирован, а тизеры построены заново. Или же, если хотите очистить все кэши (включая кэш фильтра), щелкните на кнопке Clear cached data на странице Configuration⇒Performance.

Переменные администрирования и настройки модуля

Drupal хранит большинство административных настроек в таблице `variables` и кэширует эти данные в таблице `cache`, чтобы ускорить поиск данных конфигурации. Примеры таких переменных включают название сайта, настройки для комментариев и пользователей, а также местоположение каталога `files`. Эти переменные кэшируются в единственной строке таблицы `cache`, поэтому они могут быть быстро извлечены, благодаря чему не приходится обращаться к базе данных за значением каждой переменной, которое понадобилось. Они сохраняются в виде PHP-массива, поэтому в кэше значения сериализуются, чтобы сохранить свою структуру. Любая переменная, которая использует `variable_set()` и `variable_get()` в качестве функций установки и получения значения, будет сохраняться и кэшироваться этим способом.

Мы рассмотрели мелкие и средние фрагменты, которые Drupal кэширует, чтобы оптимизировать наиболее ресурсоемкие компоненты сайта, но наибольшая оптимизация Drupal достигается при кэшировании всего представления страницы. Для анонимных пользователей это выполняется легко, поскольку все страницы выглядят одинаково. Однако для зарегистрированных пользователей каждая страница отличается от других, поскольку настроена в соответствии с их персональным профилем. Чтобы справиться с этой ситуацией, необходимо применять другую стратегию кэширования.

Для анонимных пользователей Drupal может извлечь кэшированный контент страницы посредством одного запроса, хотя для загрузки самого Drupal потребуются несколько дополнительных запросов. Соответствующие настройки находятся на странице Configuration⇒Performance административного интерфейса Drupal. Этот интерфейс приведен на рис. 16.1. Каждый из параметров настройки более подробно рассматривается в последующих разделах.

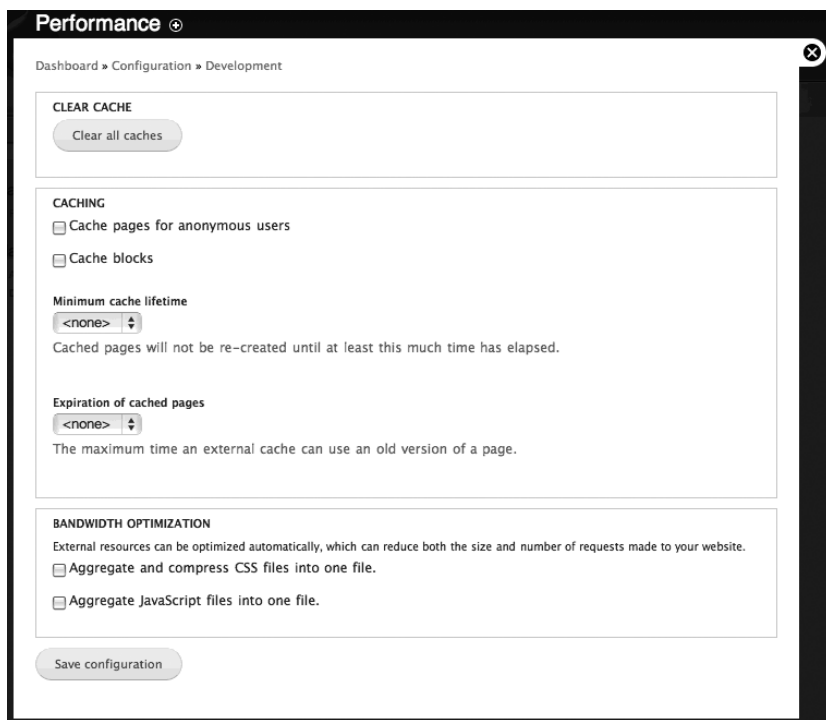


Рис. 16.1. Административный интерфейс для управления поведением кэширования страниц

Отключение кэширования

Снятие отметки с флажков Cache pages (Кэшировать страницы) и Cache blocks (Кэшировать блоки) целесообразно при отладке сайта, поскольку это позволяет видеть изменения, не очищая кэш для повторной загрузки измененных элементов. Но в общем случае кэширование должно быть включено.

На заметку! Даже если кэширование страницы отключено, Drupal все равно будет кэшировать пользовательские меню, контент фильтров, реестр тем, схему базы данных и системные переменные. Эти кэши уровня компонентов отключать нельзя.

Кэширование страниц

Кэширование страниц обеспечивает огромный рост производительности по сравнению с полным отсутствием кэширования. Это — один из самых легких способов ускорить работу медленного сайта Drupal. Давайте рассмотрим жизненный цикл запроса при включенной системе кэша страниц.

Чтобы разобраться в кэшировании страниц, вначале следует разобраться в процессе начальной загрузки Drupal. Процесс начальной загрузки состоит из маленьких, изолированных шагов, которые в Drupal называются *фазами*. Drupal получает выгоду от такой разделенной на фазы системы начальной загрузки, загружая и анализируя только тот объем кода, который необходим для обслуживания кэшированной страницы, сводя запросы к базе данных к минимуму.

Процесс обслуживания запроса кэшированной страницы для анонимного пользователя показан на рис. 16.2.

Для начала запрос вынуждает веб-сервер выполнить `index.php`. Первая строка PHP-кода в `index.php` — включение файла `includes/bootstrap.inc`, который содержит функции ядра для начальной загрузки. Затем `index.php` вызывает функцию `drupal_bootstrap()`.

Функция `drupal_bootstrap()` отвечает за выполнение всех фаз начальной загрузки. В плане кэширования для нас представляет интерес только фаза начальной загрузки `DRUPAL_BOOTSTRAP_PAGE_CACHE`. Во время нее предпринимается попытка загрузки страницы из корзины `cache_page`.

Кэширование статической страницы

По умолчанию Drupal отправляет HTTP-заголовок `Vary: Cookie` для анонимных просмотров страниц. Это указывает прокси-серверу HTTP, что тот может вернуть страницу из своего локального кэша, не обращаясь к веб-серверу, если пользователь отправляет тот же заголовок `Cookie`, что и пользователь, который первоначально запросил кэшированную страницу. Без заголовка `Vary: Cookie` аутентифицированным пользователям также предоставлялась анонимная страница из кэша. Если большинство пользователей сайта являются анонимными, за исключением нескольких известных редакторов/администраторов, заголовок `Vary` можно опустить. Это позволяет улучшить кэширование в прокси-серверах HTTP (включая инвертированные прокси-серверы), т.е., даже если клиенты отправляют различные cookie-наборы, они все еще получают контент из кэша, если агрессивное кэширование включено, а минимальное время кэша не равно нулю. Однако чтобы избежать получения кэшированных страниц от прокси-сервера, аутентифицируемые пользователи должны получать доступ к сайту непосредственно (т.е. не использовать прокси-сервер HTTP, и обходить инвертированный прокси-сервер, если таковой применяется).

Чтобы включить эту возможность, отредактируйте свой файл `default/settings.php` и удалите знак комментария из следующей строки:

```
# $conf['omit_vary_cookie'] = TRUE;
```

Блоки

Блоки можно кэшировать в зависимости от их контента. Кэширование блоков Drupal может быть включено или отключено с использованием административного интерфейса Configuration ⇒ Performance (см. рис. 16.1).

Кэширование блоков осуществляется тогда, когда модуль, представляющий блок, объявляет возможность кэширования этого блока при реагировании на операцию `list` хука `hook_block()`. Например, ниже приведен фрагмент кода реализации `hook_block_info()` из файла `modules/user/user.module`:



Рис. 16.2. Эта диаграмма демонстрирует жизненный цикл запроса анонимного пользователя, посетившего сайт, и процесс начальной загрузки Drupal, используемый для загрузки первой страницы, которую видит посетитель. На втором шаге процесса начальной загрузки Drupal пытается загрузить запрошенную страницу из кэша

```

/**
 * Реализация hook_block_info().
 */
function user_block_info() {
  global $user;

  $blocks['login']['info'] = t('User login');
  // Кэшировать не стоит.
  $blocks['login']['cache'] = DRUPAL_NO_CACHE;

  $blocks['new']['info'] = t('Who\'s new');

  // Слишком динамично для кэширования.
  $blocks['online']['info'] = t('Who\'s online');
  $blocks['online']['cache'] = DRUPAL_NO_CACHE;
  return $blocks;
}

```

В приведенном примере все блоки, предоставленные модулем user, объявляют, что они не должны кэшироваться, за одним исключением. Блок Who's new не объявляет предпочтений кэширования, а это означает, что если администратор включает кэширование блоков и затем активирует блок Who's new, он получает настройку кэширования по умолчанию DRUPAL_CACHE_PER_ROLE. Это значит, что отдельная кэшированная версия блока будет сохранена для каждой роли. Точнее говоря, отдельная кэшированная версия блока будет сохраняться для каждой комбинации ролей; идентификатор кэша создается конкатенацией идентификаторов ролей текущего пользователя (см. `_block_get_cache_id()` в `modules/block/block.module`). Возможные константы для кэширования перечислены в табл. 16.2.

Таблица 16.2. Возможные константы для установки кэширования

Константа	Значение	Смысл
DRUPAL_CACHE_CUSTOM	-2	Блок обрабатывает свой собственный кэш
DRUPAL_NO_CACHE	-1	Не кэшировать этот блок
DRUPAL_CACHE_PER_ROLE	1	Каждая роль видит отдельный кэшированный блок*
DRUPAL_CACHE_PER_USER	2	Каждый пользователь видит отдельный кэшированный блок
DRUPAL_CACHE_PER_PAGE	4	Каждая страница имеет собственный кэшированный блок
DRUPAL_CACHE_GLOBAL	8	Блоки кэшируются один раз для всех пользователей

* Установка по умолчанию для блоков, которые не объявляют настройку кэширования.

Все кэшируемые блоки кэшируются на основе темы и языка. Это предотвращает возможность наблюдения пользователями блоков, оформленных темой, отличной от применяемой пользователем, когда включена поддержка множества тем, и предотвращает отображение блока на неправильно выбранном языке при включенной многоязыковой поддержке.

На заметку! Блоки никогда не кэшируются для суперпользователя (пользователя с идентификатором 1).

Константы блока (как и константы меню) можно объединять, используя битовые операции PHP. Например, блок Book navigation, предоставленный реализацией `hook_block()` из модуля `book`, использует одновременно `DRUPAL_CACHE_PER_ROLE` и `DRUPAL_CACHE_PER_PAGE`:

```
/**
 * Реализация hook_block_info().
 */
function book_block_info() {
  $block = array();
  $block['navigation']['info'] = t('Book navigation');
  $block['navigation']['cache'] = DRUPAL_CACHE_PER_PAGE | DRUPAL_CACHE_PER_ROLE;

  return $block;
}
```

Константы `DRUPAL_CACHE_PER_ROLE` и `DRUPAL_CACHE_PER_USER` нельзя объединять битовой операцией “ИЛИ” (`|`), поскольку эти два режима кэширования являются взаимоисключающими.

Использование API-интерфейса кэширования

Разработчики модулей должны знать две функции API-интерфейса кэширования — `cache_set()` и `cache_get()`.

Кэширование данных с помощью `cache_set()`

Функция `cache_set()` используется для записи данных в кэш. Сигнатура функции такова:

```
cache_set($cid, $data, $bin = 'cache', $expire = CACHE_PERMANENT)
```

Ниже описаны параметры функции.

- `$cid`. Уникальная строка идентификатора кэша, которая действует как ключ по отношению к данным. Двоеточия используются для разделения иерархии возможностей.
- `$bin`. Имя корзины кэша, в которой хранятся данные. Допустимые значения ядра — `'cache_block'`, `'cache_bootstrap'`, `'cache_field'`, `'cache_filter'`, `'cache_form'`, `'cache_menu'`, `'cache_page'`, `'cache_update'` или `'cache'` для кэша, используемого по умолчанию.
- `$data`. Данные для сохранения в кэше. Объекты и массивы PHP автоматически сериализуются.
- `$expire`. Период времени, в течение которого кэшируемые данные остаются действительными. Возможные значения — `CACHE_PERMANENT`, `CACHE_TEMPORARY` или временная метка Unix. Если указана временная метка Unix, по истечении соответствующего ей времени данные будут трактоваться как помеченные `CACHE_TEMPORARY`.

Общий шаблон итерации для `cache_set()` можно найти в `modules/filter/filter.module`:

```
// Сохранение в кэше с минимальным сроком истечения равным 1 дню.
if ($cache) {
  cache_set($cid, $text, 'cache_filter', REQUEST_TIME + (60 * 60 * 24));
}
```

Извлечение кэшируемых данных с помощью `cache_get()` и `cache_get_multiple()`

Функция `cache_get()` служит для извлечения кэшируемых данных. Сигнатура функции выглядит следующим образом:

```
cache_get($cid, $bin = 'cache')
```

Ниже описаны параметры функции.

- `$cid`. Идентификатор кэша для извлекаемых данных.
- `$bin`. Имя корзины кэша, в которой хранятся данные. Допустимые значения ядра — `'cache_block'`, `'cache_bootstrap'`, `'cache_field'`, `'cache_filter'`, `'cache_form'`, `'cache_menu'`, `'cache_page'`, `'cache_update'` или `'cache'` для кэша, используемого по умолчанию.

Общий образец итерации для `cache_get()` можно видеть в `modules/filter/filter.module`.

```
// Проверка существования кэшированной версии данного фрагмента текста.
if ($cached = cache_get($cid, 'cache_filter')) {
    return $cached->data;
}
```

Чтобы вернуть данные из кэша для заданного массива идентификаторов кэша, необходимо воспользоваться функцией `cache_get_multiple()`. Сигнатура этой функции приведена ниже. Единственное ее отличие от функции `cache_get` состоит в том, что ей передается массив идентификаторов кэша (`$cids`).

```
cache_get_multiple(array &$cids, $bin = 'cache')
```

Использование функции `cache_is_empty` для проверки, пуст ли кэш

В ряде случаев требуется выяснить, пуста ли корзина кэша. Для проверки наличия кэшированных данных в определенной корзине можно использовать функцию `cache_is_empty`. Эта функция возвращает значение `TRUE`, если корзина кэша пуста. Сигнатура функции выглядит следующим образом:

```
cache_is_empty($bin)
```

Ниже описаны параметры функции.

- `$bin`. Имя корзины кэша, в которой хранятся данные. Допустимые значения ядра — `'cache_block'`, `'cache_bootstrap'`, `'cache_field'`, `'cache_filter'`, `'cache_form'`, `'cache_menu'`, `'cache_page'`, `'cache_update'` или `'cache'` для кэша, используемого по умолчанию.

Очистка кэша с помощью `cache_clear_all`

Поскольку вашему модулю лучше известно, когда устаревают его данные, он должен взять на себя очистку кэшей в соответствующее время. При очистке кэшей нужно руководствоваться следующими принципами.

- Очищайте наиболее специфичный кэш. Не очищайте все кэши Drupal только потому, что некоторые специфичные для модуля данные изменились! Это равносильно срыванию коврового покрытия во всем доме только потому, что требуется почистить пол на кухне.

- Используйте кэшированные данные как можно дольше. Хотя основной смысл кэширования заключается в повышении отзывчивости сайта за счет снижения объема требуемой работы, очистка кэшированных данных — достаточно трудоемкий процесс, особенно при большом их объеме.

Некоторые способы очистки кэшированных данных описаны в последующих подразделах.

Использование параметра `$reset`

У многих функций Drupal, которые выполняют внутреннее кэширование с использованием статических переменных, имеется дополнительный параметр `reset`, который очищает внутренний кэш. Например, вот уже знакомая нам функция `node_load()`:

```
node_load($nid = NULL, $vid = NULL, $reset = FALSE)
```

Третий параметр в вызове функции определяет, нужно ли выполнять сброс кэша `node_load_multiple`.

Использование `cache_clear_all()`

Главной функцией очистки кэшированных данных является `cache_clear_all()` из `includes/cache.inc`. Сигнатура этой функции выглядит следующим образом:

```
function cache_clear_all($cid = NULL, $bin = NULL, $wildcard = FALSE) {...}
```

Значения параметров `$cid` и `$bin` совпадают с их значениями для функций `cache_set()` и `cache_get()`. Параметр `$wildcard` используется для указания того, что во время очистки переданный параметр `$cid` должен трактоваться как подстрока с любым правосторонним совпадением. Ниже приведены некоторые примеры.

Очистка конкретной записи `foo:bar` из таблицы `cache`:

```
$cid = 'foo:bar';
cache_clear_all($cid, 'cache');
```

Очистка любой устаревшей записи в таблице `cache`, которая была установлено модулем `foo` (и потому имеет `$cid`, начинающийся с префикса `foo:`):

```
$cid = 'foo: '; // Будет соответствовать ключам кэша foo:bar, foo:baz и т.д.
cache_clear_all($cid, 'cache', TRUE);
```

Действительный запрос базы данных, выполняемый в предыдущем случае, выглядит так:

```
db_delete($this->bin)
  ->condition('cid', db_like($cid) . '%', 'LIKE')
  ->execute();
```

Если модуль `foo` сохраняет свои данные в собственной таблице кэша `cache_foo`, эта таблица должна быть указана, чтобы функция `cache_clear_all()` знала, что следует очищать:

```
$cid = 'foo:bar';
cache_clear_all($cid, 'cache_foo');
```

Если требуется полностью очистить таблицу кэша, нужно передать `*` в качестве `$cid` и установить параметр `$wildcard` в `TRUE`. В следующем примере очищается вся таблица `cache_foo`:

```
cache_clear_all('*', 'cache_foo', TRUE);
```

Очистка любых устаревших записей из кэшей страницы и блока (т.е. таблиц `cache_page` и `cache_block`):

```
cache_clear_all();
```

Использование `hook_flush_caches()`

В Drupal имеется центральная функция, которая очищает все кэши, включая кэши JavaScript и CSS. Это функция `drupal_flush_all_caches()` из `includes/common.inc`:

```
/**
 * Очищает все кэшированные данные на сайте.
 *
 * Опустошает таблицы кэша, перестраивает кэш меню и реестры тем, а также
 * вызывает хук, позволяющий очистить также данные кэша других модулей.
 */
function drupal_flush_all_caches() {
  // Изменение строк запросов на файлы css/js, чтобы вызвать
  // перезагрузку для всех пользователей.
  _drupal_flush_css_js();
  registry_rebuild();
  drupal_clear_css_cache();
  drupal_clear_js_cache();
  // Перестройка данных темы. Обратите внимание, что перестройка данных
  // модуля выполняется выше, как часть функции registry_rebuild().
  system_rebuild_theme_data();
  drupal_theme_rebuild();
  menu_rebuild();
  node_types_rebuild();
  // Не очищать cache_form — это может привести к прерыванию
  // выполняющейся в данный момент отправки формы.
  // Обеспечьте, чтобы очистка кэша страниц всегда была последним действием.
  $core = array('cache', 'cache_filter', 'cache_bootstrap', 'cache_page');
  $cache_tables = array_merge(module_invoke_all('flush_caches'), $core);
  foreach ($cache_tables as $table) { cache_clear_all('*', $table, TRUE); }
}
```

Если вы используете собственные таблицы кэша, функция `hook_flush_caches()` предоставляет вашему модулю шанс очистить свои кэши при щелчке на кнопке Clear cached data (Очистить кэшированные данные) на странице Configuration⇒Performance. Реализация функции `hook_flush_caches()` проста; модуль должен просто вернуть имена любых корзин кэша, которые должны быть очищены. Вот пример из модуля обновления состояния:

```
/**
 * Реализация hook_flush_caches().
 */
function examplemodule_flush_caches() {
  return array('cache_example');
}
```

В этом примере возвращается имя корзины кэша, используемой модулем examplemodule.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Знать, что собой представляют различные типы кэширования, обеспечиваемые Drupal: кэширование страницы, блока, меню, переменной и фильтра.
- Понимать, как работают системы кэширования страницы.
- Знать различия между обычным, агрессивным и кэшированием FastPath.
- Понимать, как работает система кэширования блоков.
- Уметь пользоваться функциями API-интерфейса кэширования.

ГЛАВА 17

Сеансы

НТТР является протоколом, не запоминающим состояния. Это означает, что каждое взаимодействие между веб-браузером и сервером выполняется независимо от остальных. Как же тогда отследить пользователя, который просматривает ряд веб-страниц на веб-сайте? Для этого служат сеансы. В версии PHP 4 появилась встроенная поддержка сеансов через семейство функций сеанса. В этой главе речь пойдет о том, как Drupal использует PHP-сеансы.

Что собой представляют сеансы

Сеанс — это механизм сохранения информации о посещении страниц определенным пользователем веб-сайта. Сеансы сохраняются в cookie-наборах, и им назначается уникальный идентификатор, который позволяет веб-сайту получить доступ к информации в cookie-наборе и связать эту информацию с определенным пользователем. После того как сеанс создан, ядро Drupal и соответствующие модули могут его использовать для сохранения и получения информации из cookie-наборов в процессе путешествия пользователя по сайту, не обращаясь при этом за нужными значениями к базе данных.

Drupal использует сеансы для хранения информации об аутентифицированных пользователях, а иногда и для анонимных пользователей — в тех случаях, когда модуль применяет сеансы для сохранения контекста между загрузками страницы; в противном случае сеансы для анонимных пользователей не генерируются.

Использование сеансов

Drupal внутренне использует сеансы для выполнения нескольких важных функций, связанных с хранением промежуточной информации с учетом состояния или персональных настроек отдельного пользователя. Например, функция `drupal_set_message()` должна переносить сообщения о состоянии или об ошибках со страницы, на которой произошла ошибка, на следующую страницу.

Это делается путем сохранения сообщений в массиве `messages` в сеансе пользователя:

```
/**
 * Установка сообщения, отражающего состояние выполненной операции.
 *
 * Если функция вызывается без аргументов, она возвращает
 * весь набор сообщений, не очищая их.
 *
 * @param $message
 *   Сообщение должно начинаться с прописной буквы
 *   и всегда завершаться точкой.
```

```

* @param $type
*   Тип сообщения. Допускается одно из следующих значений:
*   - 'status' (состояние)
*   - 'warning' (предупреждение)
*   - 'error' (ошибка)
* @param $repeat
*   Если это значение равно FALSE и сообщение уже установлено,
*   то сообщение не будет повторяться.
*/
function drupal_set_message($message = NULL, $type = 'status', $repeat = TRUE)
{
  if ($message) {
    if (!isset($_SESSION['messages'][$type])) {
      $_SESSION['messages'][$type] = array();
    }
    if ($repeat || !in_array($message, $_SESSION['messages'][$type])) {
      $_SESSION['messages'][$type][] = $message;
    }
    // Пометка данной страницы как не подлежащей кэшированию.
    drupal_page_is_cacheable(FALSE);
  }
  // Сообщения не устанавливаются в случае сбоя соединения с базой данных.
  return isset($_SESSION['messages']) ? $_SESSION['messages'] : NULL;
}

```

Ниже показан еще один пример из `poll.module`, в котором сеанс используется для предотвращения обращения пользователя к кэшу страницы:

```

if (!$user->uid) {
  // Результаты голосования записываются, поэтому при просмотре
  // страницы опроса пользователь получает представление результатов,
  // а не форму для голосования. Сохранение значения в переменной
  // $_SESSION оказывает полезный побочный эффект, препятствуя
  // обращению пользователя к кэшу страницы. Когда анонимное голосование
  // разрешено, кэш страницы должен содержать только форму
  // голосования, но не результаты.
  $_SESSION['poll_vote'][$node->nid] = $choice;
}

```

Drupal также использует сеансы для отслеживания загрузок файлов при предварительном просмотре ноды, для запоминания персональных настроек отображения при фильтрации списка контента сайта или списка последних журнальных записей на странице Reports ⇒ Recent log entries (Отчеты ⇒ Последние журнальные записи), а также для систем инсталляции и обновления (`install.php` и `update.php`).

Drupal создает сеансы для пользователей, которые зарегистрированы на сайте (аутентифицированных пользователей). В строке таблицы `sessions`, представляющей анонимного пользователя, столбец `uid` имеет значение 0. Поскольку сеансы специфичны для браузера (они привязаны к cookie-набору браузера), наличие нескольких браузеров, открытых на одном компьютере, ведет к созданию нескольких сеансов.

Фактические данные, сохраняемые в сеансе, сохраняются в сериализованном виде в столбце `session` таблицы `sessions`. Две строки типичной таблицы `sessions` приведены в табл. 17.1. Здесь представлены записи для суперпользователя (`uid 1`), аутентифицированного пользователя (`uid 3`) и анонимного пользователя (`uid 0`). Суперпользователь имеет настройки фильтрации регистратора (`watchdog`; используется модулем `dblog`), хранящиеся в сеансе.

Таблица 17.1. Примеры строк из типичной таблицы `sessions`

uid	sid	ssid	hostname	timestamp	cache	session
1	1ldiqzKTOnxI_ zlUs9jDlLIzlrAxcg GmM3l0oo-ux7Ws	NULL	1.2.3.4	1208464106	0	dblog_ overview_ filter _ a:0: {}
3	1WgbUe5UGP26vIY8wkxbkF A8odnqoy4x6bC7uV8lss	NULL	5.6.7.8	1208460845	0	--

Таблица `sessions` очищается при выполнении программы сборки мусора сеансов PHP. Промежуток времени, в течение которого строка остается в таблице, определяется настройкой `session.gc_maxlifetime` в `settings.php`. Если пользователь выходит из системы, строка его сеанса удаляется из базы данных немедленно. Обратите внимание, что если пользователь зашел на сайт одновременно из множества браузеров (не окон одного браузера) или с несколькими IP-адресами, для каждого браузера создается отдельный сеанс. Поэтому выход пользователя из сайта в одном браузере не означает его выхода из сайта в других браузерах.

Настройки, связанные с сеансом

Drupal изменяет настройки управления сеансами в трех местах: в файле `.htaccess`, в файле `settings.php` и в коде начальной загрузки внутри файла `includes/bootstrap.inc`.

Файл `.htaccess`

Drupal гарантирует себе полный контроль над моментом запуска сеансов, отключая PHP-функцию `session.auto_start` в файле `.htaccess`, используемом по умолчанию программой инсталляции Drupal, с помощью следующей строки:

```
php_flag session.auto_start false
```

`session.auto_start` — это опция конфигурации, которую PHP не может изменить во время выполнения, поэтому она и находится в `.htaccess`, а не в `settings.php`.

Файл `settings.php`

Большинство настроек сеансов осуществляется в файле `settings.php`, который располагается в `sites/default/settings.php` или в `sites/example.com/settings.php`:

```
ini_set('session.gc_probability', 1);
ini_set('session.gc_divisor', 100);
ini_set('session.gc_maxlifetime', 200000);
ini_set('session.cookie_lifetime', 2000000);
```

Размещение этих настроек в `settings.php`, а не в `.htaccess` позволяет подсайтам использовать другие настройки, а также дает возможность Drupal изменять настройки сеанса в хостах, выполняющих PHP как CGI (PHP-директивы в `.htaccess` в такой конфигурации не работают).

Drupal использует функцию `ini_set('session.save_handler', 'user');` для перепределения обработки сеансов по умолчанию, предоставляемой PHP, и реализации собственного управления сеансами; в этом контексте *определенный пользователем* означает “определенный Drupal” (см. www.php.net/manual/en/function.session-set-save-handler.php).

Файл bootstrap.inc

В bootstrap.inc включены следующие настройки сеанса:

```
ini_set('session.use_cookies', '1');
ini_set('session.use_only_cookies', '1');
ini_set('session.use_trans_sid', '0');

// При отправке HTTP-заголовков не использовать обработчик сеансов PHP.
ini_set('session.cache_limiter', 'none');

// Использовать только cookie-наборы http-сеанса.
ini_set('session.cookie_httponly', '1');
```

PHP предоставляет встроенные функции обработки сеансов, но позволяет переопределять эти функции, если требуется реализовать собственные обработчики, такие как хранение сеансов в модуле или в базе данных MongoDB вместо MySQL. PHP продолжает управлять cookie-наборами, в то время как реализация Drupal обрабатывает данные хранилища сеансов.

Вызов функции `drupal_session_initialize()` во время фазы `DRUPAL_BOOTSTRAP_SESSION` начальной загрузки устанавливает в качестве обработчиков функции файла `includes/sessions.inc` и запускает обработку сеанса:

```
case DRUPAL_BOOTSTRAP_SESSION:
    require_once DRUPAL_ROOT . '/' . variable_get('session_inc',
'includes/session.inc');
    drupal_session_initialize();
    break;
```

Внутри файла `session.inc` функция `drupal_session_initialize()` устанавливает обработчики в следующие значения:

```
session_set_save_handler('_drupal_session_open', '_drupal_session_close',
'_drupal_session_read', '_drupal_session_write', '_drupal_session_destroy',
'_drupal_session_garbage_collection');
```

Обратите внимание, что файл, включаемый в `bootstrap.inc`, определяется переменной Drupal по имени `session_inc`. Это означает, что можно спокойно реализовать собственную обработку сеанса и подключить ее вместо обработки сеанса Drupal по умолчанию. Например, модуль `memcache` (drupal.org/project/memcache) реализует функции `sess_open()`, `sess_close()`, `sess_read()`, `sess_write()`, `sess_destroy_sid()` и `sess_gc()`. Установка переменной `session_inc` вынуждает Drupal использовать для обработки сеансов этот код, а не обработку сеанса по умолчанию:

```
'session_inc' => './sites/all/modules/memcache/memcache-session.inc',
```

Переменную можно было бы переопределить также, устанавливая ее в своем файле `settings.php`:

```
$conf = array(
  'session_inc' => './sites/all/modules/memcache/memcache-session.inc, );...
```

Требование поддержки cookie-наборов

Если браузер не принимает cookie-наборы, установка сеанса невозможна, поскольку PHP-директива `sessions_use_only_cookies` будет иметь значение 1, а альтернативное поведение (передача `PHPSESSID` в строке запроса URL) заблокировано установкой `sessions.use_trans_sid` в 0. Zend рекомендует следующий подход (см. <http://php.net/session.configuration>).

Управление сеансами на основе URL сопряжено с дополнительными рисками безопасности по сравнению с управлением сеансами на основе cookie-наборов. Пользователи могут отправить URL, который содержит идентификатор активного сеанса, своим друзьям по электронной почте или же сохранить URL, который содержит идентификатор сеанса, в закладках браузера и, например, всегда получать доступ к сайту с одним и тем же идентификатором сеанса.

Когда PHPSESSID появляется в строке запроса сайта, обычно это служит признаком того, что поставщик хостинга заблокировал PHP и не позволяет функции `ini_set()` устанавливать директивы PHP во время выполнения. Альтернатива состоит в перемещении параметров настройки в файл `.htaccess` (если хост выполняет PHP-код в качестве модуля Apache) или в локальный файл `php.ini` (если хост выполняет PHP-код как исполняемый модуль CGI).

Для предотвращения захвата сеанса (при котором кто-либо перехватывает идентификатор сеанса из старого cookie-набора и предпринимает попытку повторного использования этого идентификатора — см. http://en.wikipedia.org/wiki/Session_hijacking), идентификатор сеанса регенерируется при регистрации пользователя (см. функцию `user_login_finalize()` в `modules/user/user.module`). Сеанс регенерируется также, когда пользователь изменяет свой пароль.

Хранилище

Информация о сеансе сохраняется в таблице `sessions`, которая связывает идентификаторы сеанса с идентификаторами пользователя Drupal в ходе выполнения фазы `DRUPAL_BOOTSTRAP_SESSION` начальной загрузки (процесс начальной загрузки Drupal более подробно описан в главе 15). Фактически объект `$user`, интенсивно используемый в Drupal, первоначально создается в ходе выполнения этой фазы функцией `_drupal_session_read()`, определенной в файле `includes/sessions.inc` (создание объекта `$user` описано в главе 6).

Структура таблицы, в которой сохраняется информация о сеансах, показана в табл. 17.2.

Таблица 17.2. Структура таблицы `sessions`

Поле	Тип	Длина	Описание
<code>uid</code>	<code>unsigned int</code>	10	Идентификатор аутентифицированного пользователя (0 — для анонимного пользователя)
<code>sid</code>	<code>varchar</code>	128	Идентификатор сеанса, сгенерированный PHP
<code>ssid</code>	<code>varchar</code>	128	Идентификатор безопасного сеанса, сгенерированный PHP
<code>hostname</code>	<code>varchar</code>	128	IP-адрес, последний использовавший этот идентификатор сеанса
<code>timestamp</code>	<code>int</code>	11	Временная метка Unix последнего запроса страницы
<code>cache</code>	<code>int</code>	11	Время последней отправки пользователя, которое используется для установки минимального времени жизни кэша
<code>сеанс</code>	<code>longblob</code>		Сериализованная версия данных, хранимых в <code>\$_SESSION</code>

Когда среда Drupal обслуживает страницу, последняя выполняемая ею задача состоит в сохранении данных сеанса в таблице `sessions` (см. `_drupal_session_write()` в `includes/session.inc`). Это делается только в случае, если браузер представляет допустимый cookie-набор, что позволяет избежать переполнения таблицы `sessions` за счет сеансов поисковых роботов.

Жизненный цикл сеанса

Жизненный цикл сеанса начинается с проверки существования cookie-набора сеанса (см. http://api.drupal.org/api/function/drupal_session_initialize/7), и если он существует, выполняется инициализация сеанса. В противном случае сеанс запускается по требованию только при необходимости сохранения чего-либо в сеансе. Этот подход позволяет анонимным пользователям просматривать сайт без использования cookie-набора сеанса, если только они не выполняют такие операции, как отправка формы, которые требуют наличия сеанса. Это позволяет Drupal предоставлять анонимным пользователям кэшированные страницы из инвертированного прокси-сервера, такого как Varnish.

Однако Drupal создает уникальный идентификатор сеанса, даже если cookie-набор сеанса не требуется, на тот случай, если сеанс все же понадобится. Примером может служить случай, когда модуль вызывает функцию `drupal_get_token()` — эта функция должна знать идентификатор сеанса заранее, еще до генерации сеанса.

Если сеанс необходим, Drupal проверяет таблицу `sessions` на предмет наличия строки с идентификатором сеанса в качестве ключа. Если такая строка найдена, функция `drupal_session_read()` из `includes/sessions.inc` извлекает данные сеанса и выполняет SQL-операцию JOIN (соединение) строки из таблицы `sessions` с соответствующей ей строкой из таблицы `users`. Результатом этого соединения будет объект, содержащий все поля и значения из обеих строк. Им является глобальный объект `$user`, который используется повсюду в Drupal (см. главу 6). Таким образом, данные сеанса доступны также посредством объекта `$user`, в частности, через `$user->session`, `$user->sid`, `$user->hostname`, `$user->timestamp` и `$user->cache`. Роли для текущего пользователя также извлекаются и присваиваются `$user->roles` в `sess_read()`.

Но что случится, если в таблице `users` нет пользователя с идентификатором, соответствующим идентификатору пользователя в сеансе? Это непростой вопрос. Поскольку инсталлятор Drupal создает строку в таблице `users` с идентификатором пользователя 0, а неаутентифицированным (анонимным) пользователям в таблице `sessions` назначается `uid` со значением 0, соединение будет работать всегда.

Внимание! Никогда не удаляйте все строки из таблицы `users` своей инсталляции Drupal. Строка, содержащая идентификатор пользователя 0, необходима для корректного функционирования самой среды Drupal.

Чтобы выяснить, когда пользователь в последний раз обращался к странице, можно просмотреть значение `$user->timestamp`, которое основано на временной метке, записанной в таблице `sessions`, либо значение `$user->access`, сохраненное в таблице `users`. Из этих двух вариантов значение `$user->timestamp`, если оно существует, обеспечит более точный результат, поскольку обновление `$user->access` в таблице `users` подвергается регулировке, чтобы запись выполнялась не чаще, чем каждые 180 секунд (по умолчанию). Это значение можно изменить, устанавливая переменную Drupal `session_write_interval` в функции `_drupal_session_write()` внутри `includes/session.inc`:


```
// Время последнего доступа обновляется не чаще, чем раз в 180 секунд.
// Это сокращает количество параллельных доступов к таблице users.
if ($user->uid && REQUEST_TIME - $user->access >
    variable_get('session_write_interval', 180)) {
    db_update('users')
    ->fields(array(
        'access' => REQUEST_TIME
    ))
    ->condition('uid', $user->uid)
    ->execute();
}
```

Естественно, ни `$user->login`, ни `$user->access` не будут существовать для пользователей, посещающих страницу впервые, или для анонимных пользователей, не имеющих сеанса, поскольку в этих случаях никакая временная метка еще не сохранена.

После того как веб-страница доставлена браузеру, последний шаг состоит в закрытии сеанса. PHP вызывает функцию `_drupal_session_write()` из `includes/session.inc`, которая записывает всю накопившуюся в `$_SESSION` информацию (в течение запроса) в таблицу `sessions`. Рекомендуется сохранять данные в `$_SESSION` только тогда, когда это действительно необходимо.

Сеансовые диалоги

Ниже представлены некоторые примеры того, что происходит с точки зрения сеансов при посещении сайта Drupal из браузера.

Первое посещение

Браузер: Я хотел бы получить страницу.

Drupal: Я могу видеть ваш cookie-набор?

Браузер: К сожалению, у меня нет cookie-набора, я посещаю сайт впервые.

Drupal: Вот страница, которую вы запрашивали.

Браузер: Мой пользователь сделал что-то, что сгенерировало сообщение.

Drupal: Хорошо, я создам сеанс и сохраню сообщение в сеансе (cookie-наборе). Вот он.

Браузер: Спасибо за cookie-набор.

Второе посещение

Браузер: Могу я получить еще одну страницу?

Drupal: Я могу видеть ваш cookie-набор?

Браузер: Вот он. В нем сообщается идентификатор сеанса `6tc47s8jd6rls9cugkdrjrm8h5`.

Drupal: Я не могу найти вас в записях. Тем не менее, вот запрошенная вами страница. Я сделаю запись о вас на случай, если вы посетите сайт снова.

Пользователь с учетной записью

(Пользователь создал учетную запись и щелкнул на кнопке `Log In` (Вход).)

Браузер: Я хотел бы получить страницу.

Drupal: Я могу видеть ваш cookie-набор?

Браузер: Вот он. В нем сообщается идентификатор сеанса 31bfa29408ebb23239042ca8f0f77652.

Drupal: Здравствуйте, Джо! Вы являетесь пользователем с идентификатором 384, предпочитаете вложенные комментарии и черный кофе. Вот новый cookie-набор, чтобы ваш сеанс не взломали. Я сделаю запись о вашем посещении. Всего хорошего.

Общие задачи

Ниже перечислены некоторые общие ситуации, при которых может потребоваться использование сеансов либо изменение настроек сеанса.

Изменение времени истечения срока действия cookie-набора

Срок действия cookie-набора, содержащего идентификатор сеанса, задается в `session.cookie_lifetime` в файле `settings.php` и по умолчанию имеет значение 2 000 000 секунд (около 23 дней). Изменение этого значения на 0 приводит к тому, что cookie-набор будет уничтожаться после закрытия браузера.

Изменение имени сеанса

Общая проблема с сеансами возникает при развертывании Drupal на нескольких поддоменах. Поскольку по умолчанию каждый сайт использует одно и то же значение для `session.cookie_domain` и одно и то же значение `PHPSESSID` для `session.name`, одновременно пользователи могут войти только на один сайт. Drupal решает эту проблему, создавая уникальное имя сеанса для каждого сайта. Имя сеанса основано на хешировании sha-256, с некоторыми модификациями, базового URL сайта.

Автоматическую генерацию имен сеансов можно обойти, убрав знак комментария из соответствующей строки в `settings.php` и задав значение переменной `$cookie_domain`. Значение должно содержать только алфавитно-цифровые символы. Соответствующий раздел `settings.php` приведен ниже.

```
/**
 * Drupal автоматически генерирует уникальное имя сеанса для каждого сайта
 * на основе его полного доменного имени. Если множество доменов указывают
 * на один и тот же сайт Drupal, можно либо перенаправить их к одному домену
 * (см. комментарий в .htaccess), либо убрать знак комментария из приведенной
 * ниже строки и указать их разделяемый базовый домен. В результате пользователи
 * останутся зарегистрированными при переходах между различными доменами.
 */
# $cookie_domain = 'example.com';
```

На заметку! В Drupal знаки комментариев в стиле Perl (#) используются только в файлах `settings.php`, `.htaccess`, `robots.txt`, а также в действительных сценариях командной оболочки, которые хранятся в каталоге `scripts`.

Хранение данных в сеансе

Хранить данные в пользовательском сеансе удобно, потому что данные автоматически сохраняются системой сеансов. Всякий раз, когда понадобится сохранить данные, которые необходимо связать с пользователем во время его визита (или множества визитов вплоть до истечения срока жизни, определенного в `session.cookie_lifetime`), следует использовать суперглобальный массив `$_SESSION`:

```
$_SESSION['favorite_color'] = $favorite_color;
```

Для извлечения этого значения при последующем запросе необходимо применять следующий код:

```
$favorite_color = $_SESSION['favorite_color'];
```

Внимание! Объект `$user` не должен использоваться для хранения информации для анонимных пользователей.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Понимать, как Drupal изменяет обработку PHP-сеанса.
- Знать, в каких файлах содержатся настройки сеанса.
- Понимать жизненный цикл сеанса и знать, как объект `$user` создается во время запроса.
- Понимать, как сохраняются и извлекаются данные из пользовательского сеанса.

ГЛАВА 18

Использование jQuery

Язык сценариев JavaScript используется повсеместно. Каждый популярный веб-браузер поставляется с интерпретатором JavaScript. Виджеты (графические элементы) инструментальной панели Apple написаны на JavaScript. Браузер Mozilla Firefox применяет JavaScript для реализации пользовательского интерфейса. На JavaScript можно писать сценарии для Adobe Photoshop. Язык JavaScript — повсюду.

Использование JavaScript прошлой версии считается дурным тоном. Если вы не в ладах с JavaScript, пора забыть прошлые обиды и приветствовать jQuery. Средство jQuery делает написание сценариев JavaScript интуитивно понятным и увлекательным занятием. Кроме того, оно является составной частью Drupal. В этой главе вы узнаете, что собой представляет jQuery и как это средство взаимодействует с Drupal. Затем мы рассмотрим практический пример.

Что собой представляет jQuery

Средство jQuery, созданное Джоном Резигом (John Resig), позволяет обойти возможные осложнения и ограничения, с которыми могут сталкиваться разработчики, имеющие дело с JavaScript. Код JavaScript громоздок и многословен, его бывает трудно ориентировать на определенные HTML- или CSS-элементы, которыми необходимо манипулировать. jQuery помогает быстро и легко найти эти элементы в документе.

Формально процесс целенаправленного обращения к какому-либо объекту называется *обходом дерева DOM* (DOM traversal). Аббревиатура DOM образована от Document Object Model (объектная модель документов). Эта модель обеспечивает древовидную структуру для доступа к элементам страницы посредством их дескрипторов и других элементов с помощью JavaScript, как показано на рис. 18.1.

На заметку! Дополнительную информацию о jQuery можно получить на официальном веб-сайте jQuery по адресу <http://jquery.com>, а также на сайте <http://www.visualjquery.com>.

При написании кода JavaScript обычно приходится тратить время на решение проблем несовместимости браузеров и операционных систем. jQuery обрабатывает это автоматически. Кроме того, в JavaScript отсутствуют многие высокоуровневые функции. В нем не решены такие общие задачи, как анимация частей страницы, перетаскивание элементов или поддержка сортируемых элементов. jQuery преодолевает и эти ограничения.

Аналогично среде Drupal, средство jQuery имеет небольшую и эффективную кодовую базу, объем которой составляет менее 30 Кбайт. В основе jQuery лежит расширяемая платформа, которая позволяет разработчикам JavaScript подключать свои модули, а на странице <http://plugins.jquery.com/> уже доступны сотни подключаемых модулей jQuery.

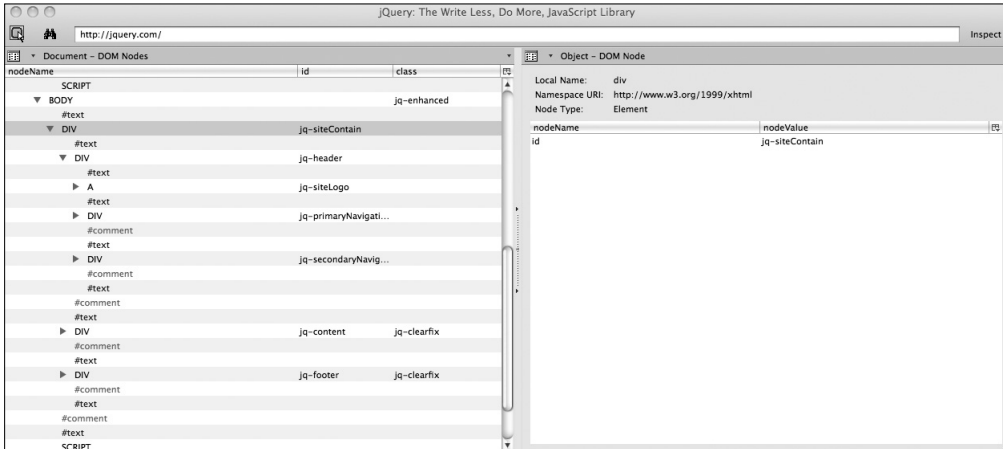


Рис. 18.1. Вывод DOM-представления <http://jquery.com> с использованием инструмента Mozilla DOM Inspector, который устанавливается вместе с браузером Firefox

Устаревший метод

Ниже для начала приведен краткий обзор естественного для JavaScript способа обхода DOM. Следующий код показывает, как система Drupal использовалась для поиска элементов на странице (в данном случае — элемента `legend` внутри всех сворачиваемых наборов полей), прежде чем появилось средство jQuery:

```
var fieldsets = document.getElementsByTagName('fieldset');
var legend, fieldset;
for (var i = 0; fieldset = fieldsets[i]; i++) {
  if (!hasClass(fieldset, 'collapsible')) {
    continue;
  }

  legend = fieldset.getElementsByTagName('legend');
  if (legend.length == 0) {
    continue;
  }
  legend = legend[0];
  ...
}
```

А вот — обновленный код в Drupal после того, как на сцене появилось средство jQuery:

```
jQuery('fieldset.collapsible >
  legend:not(.collapse-processed)', context).each(function() {
  ... });
```

Как видите, jQuery вполне соответствует своему лозунгу “Меньше кода, больше работы”. jQuery берется за общие, часто повторяющиеся задачи манипулирования DOM с помощью JavaScript и инкапсулирует его в лаконичный и интуитивно понятный синтаксис. Конечным результатом становится краткий, интеллектуальный и легко читаемый код.

Как работает jQuery

jQuery — это инструмент, предназначенный для поиска элементов в структурированных документах. Элементы документа могут выбираться с использованием селекторов CSS или собственных селекторов jQuery (подключаемый модуль jQuery поддерживает также селекторы XPath). Возможность применения селекторов CSS для обхода DOM полезна для разработчиков, поскольку большинство из них уже знакомо с синтаксисом CSS. Средство jQuery обеспечивает полную поддержку уровней CSS от 1 до 3. Прежде чем переходить к использованию jQuery в Drupal, рассмотрим несколько очень простых примеров синтаксиса jQuery.

Использование селектора идентификации CSS

Бегло рассмотрим основной синтаксис CSS. Предположим, что требуется манипулировать следующим HTML-кодом:

```
<p id="intro">Welcome to the World of Widgets.</p>
```

Для установки синего цвета фона для данного абзаца используется CSS для оформления этого конкретного абзаца в таблице стилей с применением селектора CSS идентификации `#intro`. Согласно спецификации HTML, идентификаторы должны быть уникальными в пределах данного документа, поэтому можно быть уверенным, что никакой другой элемент не обладает таким же идентификатором. Внутри таблицы стилей, которая применяется к документу, следующий фрагмент установит для абзаца синий цвет фона:

```
#intro {
  background-color: blue;
}
```

Обратите внимание, что здесь по существу выполняется две задачи: нахождение элемента с идентификатором `#intro` и установка для него синего цвета фона.

На заметку! Если вы интересуетесь работой механизма jQuery, загрузите несжатый JavaScript-код jQuery из сайта <http://jquery.com/>. Версия, включенная в Drupal, является сжатой, что уменьшает объем данных, которые браузеры вынуждены загружать с вашего сайта.

А вот как можно выбрать абзац и изменить цвет его фона на синий с помощью jQuery:

```
jQuery("#intro").css("background-color", "blue");
```

Можно даже добавить плавное появление текста абзаца:

```
jQuery("#intro").css("background-color", "blue").fadeIn("slow");
```

Использование селектора класса CSS

Ниже приведен похожий пример использования селектора класса CSS вместо селектора идентификации CSS, который рассматривался в предыдущем разделе. HTML-код выглядел бы следующим образом:

```
<p class="intro">Welcome to the World of Widgets.</p>
<p class="intro">Widgets are available in many sizes.</p>
```

Следующий код также работает, но он задает более конкретное правило:

```
p.intro {
  background-color: blue;
}
```

CSS преобразуется в код jQuery следующим образом:

```
jQuery(".intro").css("background-color", "blue").fadeIn("slow");
```

или

```
jQuery("p.intro").css("background-color", "blue").fadeIn("slow");
```

В первом из представленных выше примеров мы просим jQuery найти любой HTML-элемент, который содержит имя класса `intro`, а во втором — дескриптор абзаца, содержащий имя класса `intro`. Обратите внимание, что последний пример будет работать несколько быстрее, поскольку с учетом наложенного ограничения поиска только дескрипторов абзацев, использующих `p.intro`, jQuery приходится просматривать меньше HTML-кода.

Совет. В CSS точка (.) — это селектор класса, который может быть многократно использован в документе, а символ # является селектором уникального идентификатора элемента, имя которого может встречаться на странице только один раз.

Теперь, когда вы прочувствовали, как работает jQuery, посмотрим, как это средство взаимодействует с Drupal.

jQuery в Drupal

Использовать jQuery в рамках Drupal несложно, потому что средство jQuery устанавливается одновременно с Drupal и автоматически становится доступным при добавлении JavaScript. В Drupal файлы JavaScript добавляются функцией `drupal_add_js()` либо в файле `.info` темы. В этом разделе мы исследуем ряд элементарных функциональных возможностей jQuery в Drupal.

Ваш первый код jQuery

Итак, приступим к работе с jQuery.

1. Войдите на сайт Drupal как пользователь 1 (с учетной записью администратора).
2. На странице Modules (Модули) активируйте модуль фильтра PHP.
3. Создайте новую ноду типа Basic Page (Базовая страница), но в форме создания ноды в разделе Input format (Формат ввода) выберите PHP code (Код PHP), как показано на рис. 18.2. В качестве заголовка введите Testing jQuery (Тестирование jQuery) и добавьте следующий код в раздел body (тело) формы:

```
<?php
  drupal_add_js('jQuery(document).ready(function () {
    jQuery("p").hide();
    jQuery("p").fadeIn("slow");
  });', 'inline');
?>

<p id="one">Paragraph one</p>
<p>Paragraph two</p>
<p>Paragraph three</p>
```

4. Щелкните на кнопке Submit (Отправить) и перезагрузите страницу. Только что созданные три абзаца появятся постепенно. Неплохо, да? Обновите страницу, чтобы увидеть это еще раз. Давайте рассмотрим этот пример более подробно.

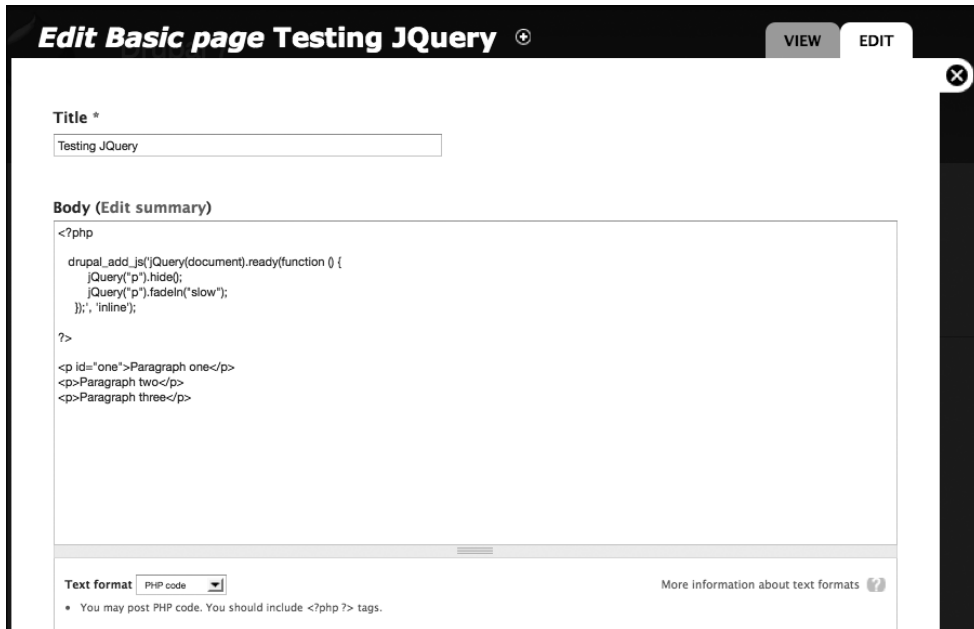


Рис. 18.2. Экспериментирование с jQuery с использованием фильтра PHP

Код jQuery содержится в файле `misc/jquery.js`. В Drupal этот файл загружается не для каждой страницы, а только при вызове функции `drupal_add_js()`. В `drupal_add_js()` передаются два параметра. Первый из них — это код JavaScript, который необходимо выполнить, а второй параметр (`inline`) заставляет Drupal помещать код между дескрипторами `<script>` и `</script>` в элементе `<head>` документа.

На заметку! Использование функции `drupal_add_js()` в этом примере достаточно просто, однако она обладает множеством других возможностей, о которых можно почитать на странице http://api.drupal.org/api/function/drupal_add_js/7.

Давайте рассмотрим JavaScript-код jQuery более подробно:

```
<?php
  drupal_add_js('jQuery(document).ready(function () {
    jQuery("p").hide();
    jQuery("p").fadeIn("slow");
  });', 'inline');
?>
```

Функция `jQuery(document).ready()` требует некоторых пояснений. Когда браузер визуализирует страницу, он попадает в точку, в которой был получен HTML-код и проведен полный синтаксический разбор DOM-структуры страницы. Следующий шаг состоит в визуализации DOM, что включает в себя загрузку дополнительных локальных — и, возможно, удаленных — файлов. При попытке выполнения кода JavaScript до того, как будет сгенерирована структура DOM, код может выдать ошибку и не выполниться, поскольку объекты, которыми он манипулирует, пока не существуют. Обычно программисты JavaScript решают эту проблему с помощью различных вариаций следующего фрагмента кода:

```
window.onload = function(){ ... }
```


Трудность использования функции `window.onload` состоит в том, что она должна ожидать загрузки дополнительных файлов, и это ожидание может затянуться. Кроме того, подход, основанный на применении `window.onload`, позволяет назначать только одну функцию. Для решения обеих проблем jQuery использует следующий простой оператор:

```
jQuery(document).ready(function() {
  // Здесь размещается ваш код.
});
```

Функция `jQuery(document).ready()` выполняется сразу после генерации DOM. По указанным причинам код jQuery следует всегда помещать в приведенный выше оператор. В JavaScript вызов `function()` определяет анонимную функцию — в данном случае функцию, содержащую код, который нужно выполнить. Остальной код достаточно очевиден:

```
// Скрытие всех абзацев.
jQuery("p").hide();

// Медленное возвращение их видимости.
jQuery("p").fadeIn("slow");
```

Предыдущий код обнаруживает все дескрипторы абзацев, скрывает их, а затем медленно отображает на странице. В терминологии jQuery `fadeIn()` называют *методом* (method). Дескриптор `p` не имеет префикса `.` или `#`, поскольку `p` является дескриптором HTML, а не классом (`.`) или идентификатором (`#`) CSS.

На заметку! Мы изменяем все дескрипторы абзаца, поэтому если вы посетите страницу списка нод, такую как `http://example.com/?q=node`, то обнаружите, что были затронуты все дескрипторы абзаца, а не только те, что находятся в тизере тестовой страницы! В примере набор дескрипторов `p`, подлежащих выбору, можно было бы ограничить, изменив файл шаблона `node.tpl.php` так, чтобы контент помещался внутрь элемента `<div class='standalone'>` при автономном отображении ноды на странице, и начав пример с `$(".standalone > p")`. Этот запрос выбирает только те элементы `p`, которые являются производными от элементов в пределах автономного класса.

Выбор элемента по идентификатору

Повторим эксперимент, но на этот раз ограничимся только первым абзацем, идентификатором которого является `one`:

```
<?php
  drupal_add_js('jQuery(document).ready(function () {
    jQuery("#one").hide();
    jQuery("#one").fadeIn("slow");
  });', 'inline');
?>

<p id="one">Paragraph one</p>
<p>Paragraph two</p>
<p>Paragraph three</p>
```

На заметку! Доступ к элементу по его идентификатору — один из самых быстрых методов селекторов jQuery, потому что он транслируется во встроенный код JavaScript: `document.getElementById("one")`. Альтернативный способ доступа, `$("#one")`, работал бы медленнее, поскольку jQuery пришлось бы искать все дескрипторы абзаца, а затем искать идентификатор `one`. Самым медленным методом селектора в jQuery является селектор класса `jQuery(".foo")`, потому что поиск нужно было бы проводить по всем элементам с классом селектора `.foo`. (В этом случае было бы быстрее выполнить запрос `jQuery("p.foo").`)

Объединение методов в цепочку

Последовательности методов jQuery можно объединять, поскольку в jQuery большинство методов возвращает объект jQuery. Давайте построим цепочку из нескольких методов и объединим их в единственной команде jQuery:

```
// Скрытие всех дескрипторов p, медленное возвращение их видимости
// и медленное перемещение текста вверх и вниз.
jQuery("p").hide().fadeIn("slow").slideUp("slow").slideDown("slow");
```

Вызовы jQuery выполняются слева направо. Предыдущий фрагмент кода находит все дескрипторы абзаца, скрывает их, затем постепенно возвращает им видимость, после чего применяет эффект скольжения для перемещения абзацев вверх и вниз. Поскольку каждый из этих методов возвращает объект-оболочку jQuery, содержащий тот же набор элементов, который он получил (все элементы p), одним и тем же набором элементов можно манипулировать снова и снова, пока не будет достигнут конечный эффект.

Добавление или удаление класса

jQuery может динамически изменять класс CSS элемента. Следующий код устанавливает для первого абзаца нашего примера красный цвет, выбирая его по идентификатору, а затем присваивая ему класс ошибки Drupal:

```
jQuery("#one").addClass("error");
```

Противоположностью addClass() является метод removeClass(). Следующий фрагмент удаляет только что добавленный класс error:

```
jQuery("#one").removeClass("error");
```

Кроме того, есть еще метод toggleClass(), который добавляет или удаляет класс при каждом вызове:

```
jQuery("#one").toggleClass("error"); // Добавляет класс error.
jQuery("#one").toggleClass("error"); // Удаляет класс error.
jQuery("#one").toggleClass("error"); // Снова добавляет класс error.
```

Помещение существующих элементов в оболочку

Вместо того чтобы просто добавить класс ошибки в элемент <p id="one">, поместим этот элемент в оболочку div, чтобы красный цвет выделялся отчетливее. Это делает следующий фрагмент кода jQuery:

```
<?php
    drupal_add_js('jQuery(document).ready(function () {
        jQuery("#one").wrap("<div class=\'error\'></div>");
    });', 'inline');
?>

<p id="one">Paragraph one</p>
<p>Paragraph two</p>
<p>Paragraph three</p>
```

Обратите внимание на отмену одиночных кавычек, которая необходима из-за применения одиночных кавычек внутри функции drupal_add_js(). Полученный результат показан на рис. 18.3.



Рис. 18.3. Абзац с идентификатором one помещен внутрь дескриптора div класса error

Изменение значений элементов CSS

jQuery может использоваться для присваивания (или повторного присваивания) значений элементам CSS. Установим сплошную рамку вокруг абзаца (рис. 18.4):

```
jQuery("#one").wrap("<div class='error'></div>").css("border", "solid");
```

Обратите внимание, что метод `css` по-прежнему воздействует на элемент `p`, а не на элемент `div`, поскольку метод `wrap` возвращает целевой элемент `p` после его помещения в оболочку.

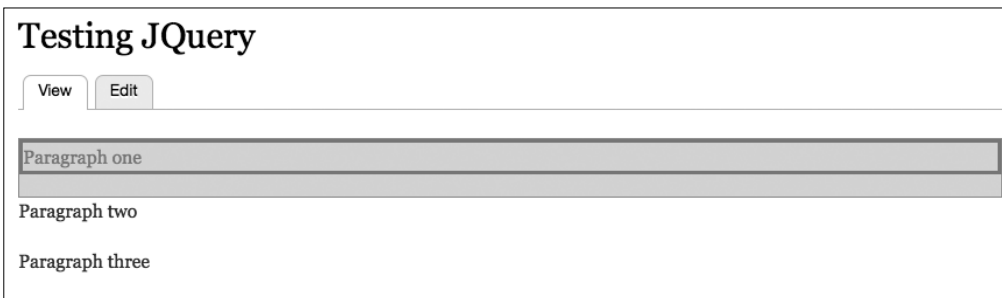


Рис. 18.4. Изменение свойства border целевого элемента

В предыдущем примере было продемонстрировано решение лишь нескольких базовых задач, которые весьма поверхностно очерчивают круг задач, решаемых с помощью jQuery. Дополнительную информацию ищите на сайте <http://jquery.com>, а также в книгах, посвященных jQuery.

Куда помещать код JavaScript

В приведенных примерах мы тестировали функции jQuery, помещая код JavaScript в код ноды с включенным фильтром PHP. Хотя это удобно для тестирования, такой подход нельзя считать удачным для рабочего сайта, где фильтр PHP рекомендуется отключать, а то и вовсе делать недоступным. Существуют другие варианты включения файлов JavaScript в разрабатываемый сайт Drupal. Например, их можно добавлять к теме, включать из модуля или даже включать их непосредственно, но предоставлять другим возможность модификации либо переопределения кода.

Добавление JavaScript посредством файла `.info` темы

Наиболее удобный, но и наименее гибкий способ включения файлов JavaScript — добавление строки в файл `.info` темы. Добавим к сайту эффект, который привлекает

внимание к логотипу, выполняя его плавное сокрытие и отображение в процессе загрузки страницы. Поместите следующий код JavaScript в файл `logofade.js` в текущей теме. Например, если применяется тема Bartik, это должен быть файл `themes/bartik/logofade.js`.

```
// Выбирает элемент темы с идентификатором logo, скрывает его,
// а затем медленно показывает снова.
jQuery(document).ready(function(){
  jQuery("#logo").fadeOut("fast").fadeIn("slow");
});
```

Файл JavaScript на месте, теперь нужно просто заставить Drupal загрузить его. Добавьте следующую строку в файл `.info` текущей темы:

```
scripts[] = logofade.js
```

Последний шаг заключается в том, чтобы заставить Drupal повторно прочитать файл `.info` и определить необходимость загрузки `logofade.js`. Для этого перейдите к станции Appearance (Внешний вид), временно переключитесь на другую тему, а затем переключитесь обратно.

Этот метод добавления JavaScript удобен, если код будет загружаться в каждой отдельной странице веб-сайта. В следующем разделе будет показано, как добавлять код JavaScript только когда активируется использующий его модуль.

Модуль, использующий jQuery

Давайте построим небольшой модуль, который включает несколько функций jQuery из файла JavaScript. Вначале нужно определиться со случаем применения. Как насчет некоторого кода JavaScript, который управляет блоками? Блоки могут быть полезны в Drupal: они могут показать состояние регистрации, сообщить о том, что нового на сайте, кто находится в онлайн, а также обеспечить удобную навигацию. Но иногда нужно просто сосредоточиться на контенте страницы! Было бы неплохо по умолчанию скрывать блоки и показывать их только тогда, когда мы хотим их видеть. Следующий модуль делает именно это, используя jQuery для идентификации и сокрытия блоков в областях левой и правой боковых панелей, и предоставляя удобную кнопку, которая вернет блоки на экран. Ниже приведен код, находящийся в файле `sites/all/modules/custom/blockaway.info`:

```
name = Block-Away
description = Uses jQuery to hide blocks until a button is clicked.
package = Pro Drupal Development
core = 7.x
files[]=blockaway.module
```

А вот код из файла `sites/all/modules/custom/blockaway.module`:

```
<?php
/**
 * @file
 * Используйте этот модуль для изучения jQuery.
 */

/**
 * Реализация hook_init().
 */
function blockaway_init() {
  drupal_add_js(drupal_get_path('module', 'blockaway') . '/blockaway.js');
}
```

Все, что делает этот модуль — включает следующий файл JavaScript, который можно поместить в `sites/all/modules/custom/blockaway/blockaway.js`:

```
/**
 * Скрытие блоков в боковых панелях, а затем их отображение по щелчку на кнопке.
 */

jQuery(document).ready(function() {
  // Получение всех элементов div класса block внутри левой боковой панели.
  // Добавление к ним всех элементов div класса block внутри
  // правой боковой панели. Проверьте файл page.tpl.php темы,
  // чтобы выяснить, какие селекторы должны использоваться —
  // следующий код соответствует теме Garland.
  var blocks = jQuery('#sidebar-first div.block, #sidebar-second div.block');

  // Скрытие этих блоков.
  blocks.hide();

  // Добавление кнопки, по щелчку на которой они вновь появятся.
  jQuery('#sidebar-first').prepend('<div id="collapsibutton">Show Blocks</div>');
  jQuery('#collapsibutton').css({
    'width': '90px',
    'border': 'solid',
    'border-width': '1px',
    'padding': '5px',
    'background-color': '#fff'
  });

  // Добавление обработчика, который запускается по щелчку на кнопке.
  jQuery('#collapsibutton').one('click', function() {
    // На кнопке совершен щелчок! Избавиться от кнопки.
    jQuery('#collapsibutton').remove();
    // Отобразить все скрытые блоки, используя эффект.
    blocks.slideDown("slow");
  });
});
```

Когда вы включаете модуль на странице Modules (Модули), любые видимые блоки должны исчезнуть, а вместо них должна отобразиться кнопка Show Blocks (Показать блоки), как показано на рис. 18.5.

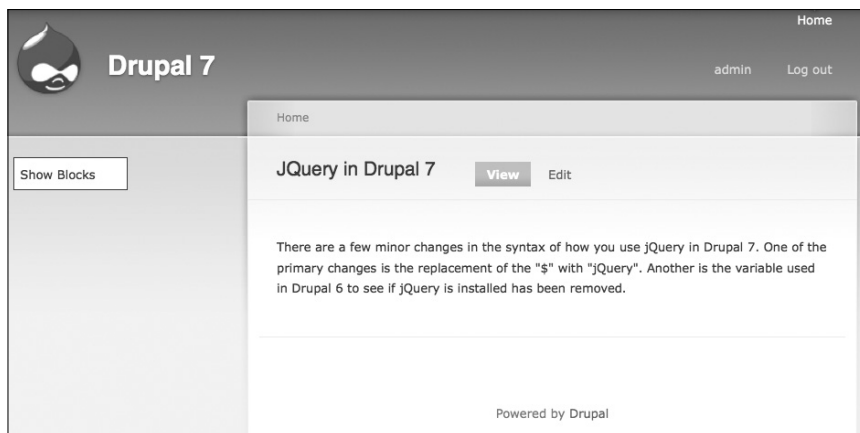


Рис. 18.5. Просмотр ноды при активированном модуле `blockaway.module`

После щелчка на этой кнопке блоки должны появиться с использованием эффекта скольжения, став видимыми (рис. 18.6).



Рис. 18.6. После щелчка на кнопке Show Blocks блоки становятся видимыми

Переопределяемый код JavaScript

Код в модуле `blockaway.module` прост и легок для понимания. Он просто проверяет факт включения файла `blockaway.js`. Однако если бы модуль был более сложным, возможно, для других разработчиков было бы удобнее, если бы вызов функции `drupal_add_js()` был помещен в функцию темы, а не в хук `hook_init()`. В результате те, кто пожелал бы использовать этот модуль, но при этом каким-либо образом изменить код JavaScript, могли бы сделать это, вообще не затрагивая кода модуля (работа системы тем описана в главе 9). Приведенный ниже код является пересмотренной версией `blockaway.module`, которая объявляет функцию темы с применением `hook_theme()`, переносит вызов `drupal_add_js()` в функцию темы и вызывает функцию темы из `hook_init()`. Функциональные возможности остаются теми же, но теперь сообразительные разработчики могут переопределять файл `blockaway.js`.

```
<?php
/**
 * @file
 * Используйте этот модуль для изучения jQuery.
 */

/**
 * Реализация hook_init().
 */
function blockaway_init() {
  theme('blockaway_javascript');
}

/**
```

```

* Реализация hook_theme().
* Регистрация функции темы.
*/
function blockaway_theme() {
  return array(
    'blockaway_javascript' => array(
      'arguments' => array(),
    ),
  );
}
/**
* Функция темы, которая только проверяет,
* включен ли наш файл JavaScript.
*/
function theme_blockaway_javascript() {
  drupal_add_js(drupal_get_path('module', 'blockaway') . '/blockaway.js');
}

```

Посмотрим, работает ли этот подход. Мы собираемся переопределить код JavaScript, предоставляемый модулем, кодом JavaScript, который предоставлен темой. Скопируйте sites/all/modules/custom/blockaway/blockaway.js в текущую тему, например, themes/bartik/blockaway.js. Теперь необходимо немного изменить файл JavaScript, чтобы было понятно, какой файл JavaScript используется. Измените эффект `slideDown("slow")` на `fadeIn(5000)`; это приведет к постепенному появлению блоков в течение пяти секунд. Код в новом файле имеет следующий вид:

```

/**
* Скрытие блоков в боковых панелях, а затем их отображение по щелчку на кнопке.
*/
jQuery(document).ready(function() {
  // Получение всех элементов div класса block внутри левой боковой панели.
  // Добавление к ним всех элементов div класса block внутри
  // правой боковой панели.
  var blocks = jQuery('#sidebar-first div.block, #sidebar-second div.block');
  // Скрытие этих блоков.
  blocks.hide();

  // Добавление кнопки, по щелчку на которой они вновь появятся.
  jQuery('#sidebar-first').prepend('<div id="collapsibutton">Show Blocks
    </div>');
  jQuery('#collapsibutton').css({
    'width': '90px',
    'border': 'solid',
    'border-width': '1px',
    'padding': '5px',
    'background-color': '#fff'
  });

  // Добавление обработчика, который запускается по щелчку на кнопке.
  jQuery('#collapsibutton').one('click', function() {
    // На кнопке совершен щелчок! Избавиться от кнопки.
    jQuery('#collapsibutton').remove();
    // Отобразить все скрытые блоки, используя эффект.
    blocks.fadeIn("5000");
  });
});

```

Последнее изменение направлено на то, чтобы заставить Drupal загрузить этот новый файл JavaScript вместо файла из `sites/all/modules/custom/blockaway`. Мы сделаем это, переопределив функцию темы. Добавьте следующую функцию в файл `template.php` своей темы (если тема не имеет файла `template.php`, создайте его):

```
/**
 * Переопределение theme_blockaway_javascript() следующей функцией.
 */
function bartik_blockaway_javascript() {
  drupal_add_js(path_to_theme() . '/blockaway.js');
}
```

На заметку! Измените имя функции предварительной обработки, чтобы в нем присутствовало имя используемой темы. В приведенном примере применялась тема Bartik.

Зайдите на страницу Modules и перестройте реестр тем, чтобы внесенные изменения были распознаны. При посещении страницы в браузере на ней должна отображаться кнопка Show Blocks (Показать блоки), щелчок на которой приведет к плавному появлению блоков вместо эффекта скольжения, использованного ранее. Итак, теперь вы знаете, как использовать код jQuery в своем модуле, как писать его в дружественном к темам и другим разработчикам стиле, а также переопределять или расширять файлы JavaScript, предоставляемые другими разработчиками модулей.

Но прежде чем завершить рассмотрение этого примера, давайте посмотрим, как можно переопределять файл шаблона. Для начала удалите функцию `bartik_blockaway_javascript()`, которую ранее добавили в файл `template.php`. Затем в текущей теме создайте пустой файл по имени `blockaway-javascript.tpl.php`. Например, если применяется тема Bartik, создайте файл `themes/bartik/blockaway-javascript.tpl.php`. Ничего не помещайте в этот файл. Теперь зайдите на страницу Modules. Посещение этой страницы вызовет перестройку реестра тем. Drupal найдет файл шаблона и использует его вместо функции темы из вашего модуля. В результате `blockaway.js` никогда не будет загружаться; фактически это равносильно комментированию функции темы за счет создания пустого файла шаблона (вспомните из главы 9, что при построении реестра тем Drupal ищет сначала файл шаблона, а потом — функции темы).

Теперь добавьте следующий код в файл `blockaway-javascript.tpl.php`:

```
<?php drupal_add_js(path_to_theme() . '/blockaway.js'); ?>
```

После перегрузки страницы должно быть видно, что теперь загружается файл JavaScript. Эти приемы могут оказаться весьма полезными, когда необходимо использовать собственный расширенный файл JavaScript в модуле от независимого разработчика либо для предотвращения загрузки какого-либо кода JavaScript.

На заметку! Функцию `drupal_add_js()` нельзя вызывать из `page.tpl.php` или любой функции темы, вызываемой при предварительной обработке (например, в блоках), потому что они выполняются слишком поздно в ходе процесса построения страницы. Пример базового шаблонного файла, добавляющего код JavaScript, можно найти в `modules/block/block-admin-display-form.tpl.php`.

Создание виджета для голосования с помощью jQuery

Давайте создадим более сложный модуль Drupal, поддерживающий jQuery. Мы построим виджет (графический элемент) для голосования с использованием AJAX (рис. 18.7), который позволяет пользователям добавлять один голос к понравившемуся сообщению. Мы применим jQuery для добавления голосов, а также для изменения общего количества голосов без перезагрузки всей страницы. Мы добавим также основанное на ролях разрешение `rate content` (оценить контент), которое позволит участвовать в голосовании только пользователям, обладающим этим разрешением. Поскольку в ходе голосования пользователи могут добавить только один голос, назовем модуль `plusone`.

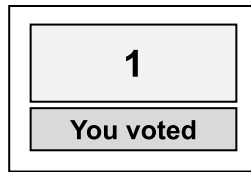


Рис. 18.7. Виджет для голосования

Прежде чем приступить к созданию самого кода jQuery модуля `plusone`, придется построить некоторый базовый модуль. Если вам никогда не приходилось создавать модули, обратитесь к главе 2. В противном случае можно сразу приступить к работе.

Создайте каталог в каталоге `sites/all/modules/custom` и назовите его `plusone` (возможно, вначале придется создать каталог `sites/all/modules/custom`). В каталоге `plusone` создайте файл `plusone.info`, содержащий следующие строки:

```
name = Plus One
description = "A +1 voting widget for nodes. "
package = Pro Drupal Development
core = 7.x
files[]=plusone.module
```

Этот файл регистрирует модуль в Drupal, чтобы его можно было активировать или отключать в административном интерфейсе.

Затем понадобится создать файл `plusone.install`. Функции, содержащиеся в этом PHP-файле, вызываются, когда модуль активируется, отключается, устанавливается или деинсталлируется, обычно для создания или удаления таблиц из базы данных. В данном случае мы будем отслеживать, кто за какую ноду голосовал.

```
<?php
/**
 * Реализация hook_install().
 */
function plusone_install() {
  // Создание таблиц.
  drupal_install_schema('plusone');
}
/**
 * Реализация hook_schema().
 */
function plusone_schema() {
  $schema['plusone_votes'] = array(
    'description' => t('Stores votes from the plusone module.'),
```

```

'fields' => array(
  'uid' => array(
    'type' => 'int',
    'not null' => TRUE,
    'default' => 0,
    'description' => t('The {user}.uid of the user casting the vote.'),
  ),
  'nid' => array(
    'type' => 'int',
    'unsigned' => TRUE,
    'not null' => TRUE,
    'default' => 0,
    'description' => t('The {node}.nid of the node being voted on.'),
  ),
  'vote_count' => array(
    'type' => 'int',
    'not null' => TRUE,
    'default' => 0,
    'description' => t('The number of votes cast.'),
  ),
),
'primary key' => array('uid', 'nid'),
'indexes' => array(
  'nid' => array('nid'),
  'uid' => array('uid'),
),
);
return $schema;
}

```

Добавьте также файл `sites/all/modules/custom/plusone/plusone.css`. Этот файл не является абсолютно необходимым, однако он несколько улучшит внешний вид виджета для голосования (рис. 18.8).

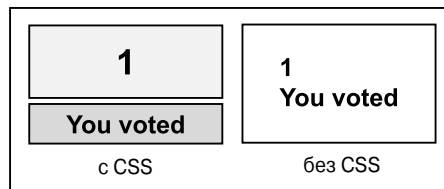


Рис. 18.8. Виджет для голосования с применением CSS и без него

Добавьте в файл `plusone.css` следующий код:

```

div.plusone-widget {
  width: 100px;
  margin-bottom: 5px;
  text-align: center;
}
div.plusone-widget .score {
  padding: 10px;
  border: 1px solid #999;
  background-color: #eee;
  font-size: 175%;
}

```

```
div.plusone-widget .vote {
  padding: 1px 5px;
  margin-top: 2px;
  border: 1px solid #666;
  background-color: #ddd;
}
```

Теперь, когда вспомогательные файлы созданы, сосредоточим внимание на файлах модуля и JavaScript jQuery. Создайте два пустых файла: `sites/all/modules/custom/plusone/plusone.js` и `sites/all/modules/custom/plusone/plusone.module`. В последующих нескольких шагах мы будем постепенно добавлять код в эти файлы. В конечном счете, должны присутствовать следующие файлы:

```
sites/
  all/
    modules/
      custom/
        plusone/
          plusone.js
          plusone.css
          plusone.info
          plusone.install
          plusone.module
          plusone-widget.tpl.php
```

Построение модуля

Откройте пустой файл `plusone.module` в текстовом редакторе и поместите в него стандартный заголовок документирования Drupal:

```
<?php
/**
 * @file
 * Простой виджет для голосования +1.
 */
```

Теперь можно приступать к написанию хуков Drupal. Один из простых хуков — `hook_permissions()` — позволяет добавить разрешение `rate content` в страницу управления доступом Drupal на основе ролей. Это разрешение используется, чтобы воспрепятствовать голосовать анонимным пользователям без предварительного создания учетной записи или входа на сайт.

```
/**
 * Реализация hook_permission().
 */
function plusone_permission() {
  $perms = array(
    'rate content' => array(
      'title' => t('Rate content'),
    ),
  );
  return $perms;
}
```

Теперь пора приступить к реализации ряда функциональных возможностей AJAX. Одна из замечательных особенностей jQuery заключается в возможности отправлять собственные HTTP-запросы GET или POST, благодаря чему можно отправить Drupal свои данные голосования, не обновляя при этом всю страницу. jQuery будет перехватывать

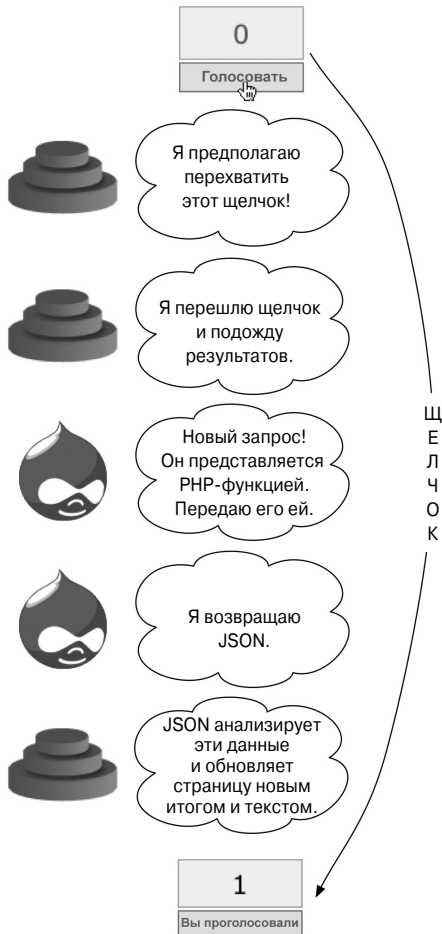


Рис. 18.9. Процесс обновления результатов голосования

щелчок на ссылке **Vote** (Голосовать) и отправлять Drupal запрос на сохранение голоса и возвращение обновленного итога. Затем jQuery использует новое значение для обновления результатов голосования на странице. Общая картина процесса показана на рис. 18.9.

После перехвата щелчка на ссылке **Vote** jQuery нуждается в вызове функции Drupal посредством URL. Для отображения URL голосования, отправленного jQuery, на PHP-функцию Drupal будет использоваться `hook_menu()`. PHP-функция сохраняет голос в базе данных и возвращает jQuery новое значение итога в виде данных JSON (JavaScript Object Notation — объектная нотация JavaScript). (Здесь не используется XML, поэтому в строгом смысле это не чистый AJAX.)

```
/**
 * Реализация hook_menu().
 */
function plusone_menu() {
  $items['plusone/vote'] = array(
    'title' => 'Vote',
    'page callback' => 'plusone_vote',
    'access arguments' =>
      array('rate content'),
    'type' => MENU_SUGGESTED_ITEM,
  );
  return $items;
}
```

В представленной выше функции поступающий запрос для пути `plusone/vote` обрабатывается функцией `plusone_vote()`, если пользователь, запрашивающий путь, имеет разрешение `rate content`.

На заметку! Если пользователь, осуществляющий этот вызов, не имеет разрешения `rate content`, Drupal вернет страницу **Access Denied** (Доступ запрещен). Это значит, что виджет для голосования нужно строить динамически, чтобы те, кому не разрешено голосовать, вообще не видели ссылку голосования. Обратите внимание на то, как система разрешений Drupal защищает от злоумышленников, которые могут попытаться обойти виджет и непосредственно обратиться к URL `http://example.com/?q=plusone/vote`.

Путь `plusone/vote/3` преобразуется в вызов PHP-функции `plusone_vote(3)` (система меню/обратных вызовов рассматривается в главе 4):

```
/**
 * Вызывается кодом jQuery или браузером, если JavaScript отключен.
 * Отправляет запрос на голосование. Если вызывается кодом jQuery,
 * возвращает JSON. Если вызывается браузером, возвращает страницу
 * с обновленным итогом голосования.
 */
```

```

function plusone_vote($nid) {
    global $user;
    $nid = (int)$nid;

    // Авторы не могут голосовать за собственные сообщения.
    // Проверяем таблицу node, чтобы посмотреть, не является
    // ли пользователь автором сообщения.
    $is_author = db_query('SELECT uid from {node} where nid = :nid AND uid = :uid',
        array(":nid" => (int)$nid, ":uid" => (int)$user->uid))->fetchField();

    if ($nid > 0 && !$is_author) {
        // Получение текущего счетчика голосов для данного пользователя.
        $vote_count = plusone_get_vote($nid, $user->uid);
        echo "Vote count is: $vote_count<br/>";
        if (!$vote_count) {
            echo "Yep was existing votes<br/>";
            // Удаление существующего счетчика голосов для данного пользователя.
            db_delete('plusone_votes')
                ->condition('uid', $user->uid)
                ->condition('nid', $nid)
                ->execute();
            db_insert('plusone_votes')
                ->fields(array(
                    'uid' => $user->uid,
                    'nid' => $nid,
                    'vote_count' => $vote_count + 1,
                ))
                ->execute();
        }
    }
    $total_votes = plusone_get_total($nid);
    // Проверка того, выполнен ли вызов кодом jQuery. Вызов AJAX использует
    // метод POST и передает пару "ключ/значение" вида js = 1.
    if (!empty($_POST['js'])) {
        // Вызов был выполнен кодом jQuery.
        // Этот код вернет результаты запросу jQuery.
        drupal_json(array(
            'total_votes' => $total_votes,
            'voted' => t('You Voted')
        ));
        exit();
    }

    // Этот вызов был выполнен не кодом JavaScript.
    // Нужно заново отобразить всю страницу с обновленным
    // результатом голосования, перенаправив ответ node/$nid
    // (или любому псевдониму URL, который был установлен для node/$nid).
    $path = drupal_get_path_alias('node/'. $nid);
    drupal_goto($path);
}

```

Приведенная функция `plusone_vote()` сохраняет текущий голос и возвращает jQuery информацию в виде ассоциативного массива, содержащего новое значение итога и строку **You voted** (Вы проголосовали), которая заменяет строку **Vote** (Голосовать) под виджетом для голосования. Этот массив передается в функцию `drupal_json()`, которая

преобразует PHP-переменные в их JavaScript-эквиваленты, в данном случае преобразуя ассоциативный массив PHP в объект JavaScript, и устанавливает HTTP-заголовок в Content-type: text/javascript. Описание формата JSON можно найти по адресу <http://ru.wikipedia.org/wiki/JSON>.

Следует отметить, что предыдущая функция реализована не особенно изящно. При записи кода jQuery необходимо обеспечить, чтобы вызов AJAX из jQuery передавался с параметром js, а для отправки применялся метод POST. Если параметр js отсутствует, значит, пользователь щелкнул на ссылке Vote и браузер сам запросил путь, например, plusone/vote/3. В этом случае при возврате данных формат JSON не используется, поскольку браузер ожидает обычную HTML-страницу. Вместо этого итог голосования обновляется для отражения факта голосования пользователя, после чего браузер переадресуется к первоначальной странице, которая будет перестроена Drupal и отобразит новое значение итога.

В приведенном коде вызывались функции `plusone_get_vote()` и `plusone_get_total()`, поэтому создадим их:

```
/**
 * Возврат числа голосов для данной пары идентификатор
 * ноды/идентификатор пользователя.
 */
function plusone_get_vote($nid, $uid) {
  $vote_count = db_query('SELECT vote_count FROM {plusone_votes} WHERE
    nid = :nid AND uid = :uid', array(':nid' => $nid,
    ':uid' => $uid))->fetchField();
  return $vote_count;
}
/**
 * Возврат общего числа голосов для данной ноды.
 */
function plusone_get_total($nid) {
  $total_count = db_query('SELECT SUM(vote_count) from {plusone_votes}
where nid = :nid',
array(':nid' => $nid));
  return ($total_count);
}
```

Теперь сосредоточим внимание на отображении виджета для голосования рядом с сообщениями. Эта задача состоит из двух частей. Вначале соберем информацию, необходимую для отображения виджета в `hook_node_load()`:

```
/**
 * Загрузка значений, необходимых для работы виджета,
 * и вывод виджета при вызове hook_node_load.
 */
function plusone_node_view($node, $view_mode) {
  global $user;

  $total = plusone_get_total($node->nid);
  $is_author = db_query('SELECT uid from {node} where nid = :nid AND uid = :uid',
    array(":nid" => $node->nid, ":uid" => $user->uid))->fetchField();

  if ($is_author) {
    $is_author = TRUE;
  } else {
    $is_author = FALSE;
  }
}
```

```

$voted = plusone_get_vote($node->nid, $user->uid);

if ($view_mode == 'full') {
  $node->content['plusone_vote'] = array(
    '#markup' => theme('plusone_widget', array('nid' => (int)$node->nid,
    'total' => (int)$total, 'is_author' => $is_author, 'voted' => $voted)),
    '#weight' => 100,
  );
  return $node;
}
}

```

Мы должны будем создать сценарий JavaScript/jQuery, который обработает щелчки пользователей на кнопке голосования и вызовет соответствующую функцию в модуле plusone для записи голоса пользователя. Этот сценарий JavaScript добавляет к ссылке a.plusone-link слушатель событий (вспомните, что plusone-link определена как селектор класса CSS), поэтому, когда пользователи щелкают на ссылке, он отправляет HTTP-запрос POST по указанному URL. Приведенный код также демонстрирует, каким образом jQuery может передавать данные обратно в Drupal. После завершения запроса AJAX возвращаемое значение (отправленное из Drupal) передается в качестве параметра data в анонимную функцию, присвоенную переменной voteSaved. Ссылка на массив выполняется посредством ключей ассоциативного массива, которые были первоначально построены функцией plusone_vote() внутри Drupal. И, наконец, сценарий JavaScript обновляет итоги голосования и изменяет текст Vote (Голосовать) на You voted (Вы проголосовали). Чтобы предотвратить перезагрузку всей страницы (поскольку щелчок обрабатывается сценарием JavaScript), возвращаемым значением функции JavaScript jQuery должно быть false.

В каталоге модуля plusone понадобится создать файл plusone.js, содержащий следующий код:

```

// Запустить следующий код, когда DOM будет полностью загружена.
jQuery(document).ready(function () {
  // Присоединить некоторый код к событию щелчка
  // на ссылке с классом plusone-link.
  jQuery('a.plusone-link').click(function () {
    // При щелчке сначала нужно определить анонимную
    // функцию, присвоенную переменной voteSaved.
    var voteSaved = function (data) {
      // Обновление количества голосов.
      jQuery('div.score').html(data.total_votes);
      // Обновление строки "Vote" строкой "You voted".
      jQuery('div.vote').html(data.voted);
    }
    // Выполнение вызова AJAX; в случае успеха
    // анонимная функция в voteSaved выполняется.
    jQuery.ajax({
      type: 'POST', // Использование метода POST.
      url: this.href,
      dataType: 'json',
      success: voteSaved,
      data: 'js=1' // Передача пары "ключ/значение".
    });
    // Предотвращение обработки щелчка браузером.
    return false;
  });
});

```

И, наконец, создадим файл `pluseone-widget.tpl.php` в каталоге модуля `pluseone`. Этот файл будет содержать следующий код:

```
<?php
/**
 * @file
 * Шаблон для отображения виджета для голосования.
 */

// Добавление файлов JavaScript и CSS.
drupal_add_js(drupal_get_path('module', 'pluseone') . '/pluseone.js');
drupal_add_css(drupal_get_path('module', 'pluseone') . '/pluseone.css');

// Построение структуры вывода.
$output = '<div class="pluseone-widget">';
$output .= '<div class="score">'. $total . '</div>';
$output .= '<div class="vote">';

// Отображение соответствующей метки под счетчиком голосов,
// исходя из значений атрибутов.
if ($is_author) {
  // Пользователь — автор. Голосовать нельзя.
  $output .= t('Votes');
}
elseif ($voted > 0) {
  // Пользователь уже голосовал. Повторное голосование запрещено.
  $output .= t('You voted');
}
else {
  // Пользователь имеет право голосовать.
  $output .= l(t('Vote'), "pluseone/vote/$nid", array(
    'attributes' => array('class' => 'pluseone-link')
  ));
}

$output .= '</div>'; // Закрытие элемента div с классом vote.
$output .= '</div>'; // Закрытие элемента div с классом pluseone-widget.

print $output;
```

В приведенном коде мы использовали переменные, установленные функции `hook_node_load` в файле `pluseone-widget.tpl.php`, что позволяет отобразить виджет. Создание отдельного шаблона темы вместо построения HTML-кода внутри самого модуля позволяет дизайнерам переопределять эту функцию, если они хотят изменить разметку.

HTML-код виджета, который появится на странице `http://example.com/?q=node/4`, должен выглядеть примерно так:

```
<div class="pluseone-widget">
  <div class="score">0</div>
  <div class="vote">
    <a class="pluseone-link" href="/pluseone/vote/4">Vote</a>
  </div>
</div>
```

Использование `Drupal.behaviors`

Взаимодействие с JavaScript осуществляется за счет присоединения поведений (т.е. действий, инициируемых событиями, такими как щелчок кнопкой мыши) к элементам в

DOM. Изменение в DOM может привести к утере этой привязки. Поэтому хотя использованный ранее файл `plusone.js` может успешно работать с базовым сайтом Drupal, он может столкнуться с проблемами, если другие файлы JavaScript станут манипулировать моделью DOM. Drupal предоставляет центральный объект `Drupal.behaviors`, с помощью которого функции JavaScript могут регистрироваться для реализации повторных привязок поведений при необходимости. Следующая версия `plusone.js` позволяет реализовать голосование с помощью AJAX в точности как предыдущая версия, но сохраняет нашу привязку, выполняя регистрацию с помощью `Drupal.behaviors`:

```
Drupal.behaviors.plusone = function (context) {
  jQuery('a.plusone-link:not(.plusone-processed)', context)
    .click(function () {
      var voteSaved = function (data) {
        jQuery('div.score').html(data.total_votes);
        jQuery('div.vote').html(data.voted);
      }
      jQuery.ajax({
        type: 'POST',
        url: this.href,
        dataType: 'json',
        success: voteSaved,
        data: 'js=1'
      });
      return false;
    })
    .addClass('plusone-processed');
}
```

Чтобы подробнее ознакомиться с `Drupal.behaviors`, обратитесь к `misc/drupal.js`.

Способы расширения модуля

Замечательным расширением этого модуля было бы предоставление администратору сайта возможности активировать виджет для голосования только для нод определенных типов. Этого можно достичь так же, как было сделано для модуля аннотирования нод, построенного в главе 2. В этом случае перед добавлением виджета нужно проверить в `hook_node_view()`, разрешено ли голосование для данного типа ноды. Существует масса других возможных расширений, наподобие взвешенных голосований на основе ролей или ограничения пользователя определенным количеством голосований в течение суток. Однако наша задача состояла в том, чтобы оставить модуль простым, который бы лишь подчеркивал способы взаимодействия между Drupal и jQuery.

Совместимость

Информацию по совместимости jQuery, а также о собственно jQuery, можно найти по адресу <http://docs.jquery.com>. Если кратко, то jQuery поддерживается в следующих браузерах:

- Internet Explorer 6.0 и последующих версий
- Mozilla Firefox 1.5 и последующих версий
- Apple Safari 2.0.2 и последующих версий
- Opera 9.0 и последующих версий

Более подробные сведения о совместимости браузеров доступны по адресу http://docs.jquery.com/Browser_Compatibility.

Дальнейшие шаги

Чтобы узнать больше о том, как Drupal использует jQuery, загляните в каталог `misc` инсталляции Drupal. Там вы найдете файлы JavaScript, отвечающие за автоматическое заполнение полей форм, пакетную обработку, сворачиваемость наборов полей, создание индикаторов хода работ, перетаскиваемые строки таблиц и многое другое. Почитайте также статьи на форуме Drupal JavaScript Group по адресу <http://groups.drupal.org/javascript>.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Знать, что собой представляет jQuery.
- Понимать основные концепции работы jQuery.
- Уметь включать файлы JavaScript в разрабатываемый модуль.
- Знать, как jQuery и Drupal взаимодействуют при обмене запросами и данными.
- Уметь построить простой виджет для голосования.

ГЛАВА 19

Локализация и перевод

Локализация — это замена строк в пользовательском интерфейсе переведенными строками в соответствии с локалью пользователя. Среда Drupal разработана и используется международным сообществом. Поэтому Drupal по умолчанию поддерживает локализацию наряду с поддержкой тем для языков с письмом справа налево, таких как арабский и иврит. В этой главе будет показано, как включить локализацию и как использовать перевод интерфейса для выборочной замены встроенных строк Drupal переведенными. Затем будут рассмотрены готовые переводы, и вы узнаете, как их создавать, импортировать и экспортировать. И, наконец, будет исследована способность Drupal представлять один и тот же контент на нескольких языках (например, в канадском веб-сайте, представляющем контент на французском и английском языках), а также то, как Drupal выбирает соответствующий язык для отображения.

Активация модуля locale

Модуль локализации locale, который предоставляет функциональные возможности языка и перевода пользовательского интерфейса в Drupal, в новой инсталляции Drupal по умолчанию отключен. Это соответствует философии Drupal — предоставлять функциональные возможности только по мере необходимости. Активировать модуль локализации можно на странице Modules (Модули). Если система Drupal инсталлирована с использованием языка, отличного от английского, модуль локализации locale активируется еще во время процесса инсталляции. В примерах, приведенных в этой главе, предполагается, что этот модуль активирован.

Перевод пользовательского интерфейса

Интерфейс Drupal состоит из слов, фраз и предложений, посредством которых осуществляется взаимодействие с пользователем. В следующем разделе будет показано, как это все можно изменить. С учетом того, что в основе перевода лежит замена строк текста, в примерах основное внимание будет уделено замене строк.

Строки

С точки зрения программирования строка — это последовательность символов, такая как пятисимвольная строка Hello. Перевод строк образует основу перевода пользовательского интерфейса в Drupal. Когда среда Drupal подготавливает строку к выводу, она проверяет, требует ли строка перевода, в результате чего, если включен английский язык, отображается слово “Hello”, а если французский — то “Bonjour”. Давайте посмотрим, как это происходит.

Перевод строк с помощью функции `t()`

Все строки в Drupal, которые будут отображены конечному пользователю, должны обрабатываться функцией `t()`; это — функция *перевода* (translate) Drupal, имя которой для удобства сокращено до `t`, поскольку она используется очень часто.

На заметку! В некоторых местах Drupal функция `t()` запускается неявно, например, в строках, переданных функции `watchdog()`, или в заголовках и описаниях в хуке меню. Формы множественного числа переводятся функцией `format_plural()`, которая самостоятельно заботится о вызове функции `t()` (см. http://api.drupal.org/api/function/format_plural/7).

Специфичная для локализации часть этой функции выглядит следующим образом:

```
function locale($string = NULL, $context = NULL, $langcode = NULL) {
  global $language;
  $locale_t = &drupal_static(__FUNCTION__);

  if (!isset($string)) {
    // Если никакая строк не была указана, возврат всех кэшированных строк.
    return $locale_t;
  }

  $langcode = isset($langcode) ? $langcode : $language->language;

  // Код, который извлекает переводы из кэша или базы данных,
  // для краткости удален из этого примера.

  return ($locale_t[$langcode][$context][$string] ===
    TRUE ? $string : $locale_t[$langcode][$context][$string]);
}
```

Помимо перевода, функция `t()` также обрабатывает вставку значений в заполнители внутри строк. Обычно эти значения являются вводимыми пользователем, и они должны быть подвергнуты текстовому преобразованию, прежде чем будут отображены.

```
t('Hello, my name is %name.', array('%name' => 'John'));

Hello, my name is John.
```

Место для размещения текста, который будет вставлен, обозначается заполнителями, а текст, который должен быть вставлен, находится в снабженном ключами массиве. Этот процесс преобразования текста важен с точки зрения безопасности Drupal (дополнительную информацию ищите в главе 21). Процесс обработки перевода функцией `t()` показан на рис. 19.1, а позже, на рис. 21.1, вы увидите, как эта функция обрабатывает заполнители.

Замена встроенных строк специальными строками

Перевод пользовательского интерфейса в сущности сводится к замене одних строк другими. Начнем с малого, выбрав для замены только несколько строк. Существует несколько возможных решений задачи перевода. Мы подойдем к ознакомлению с ними, начав с самого простого способа и закончив наиболее сложным. Первый способ подразумевает редактирование файла настроек, а второй — применение модуля локализации. Начнем с выполнения простой замены строки в навигационной цепочке и заменим в ней *Blog* на *Journal*.

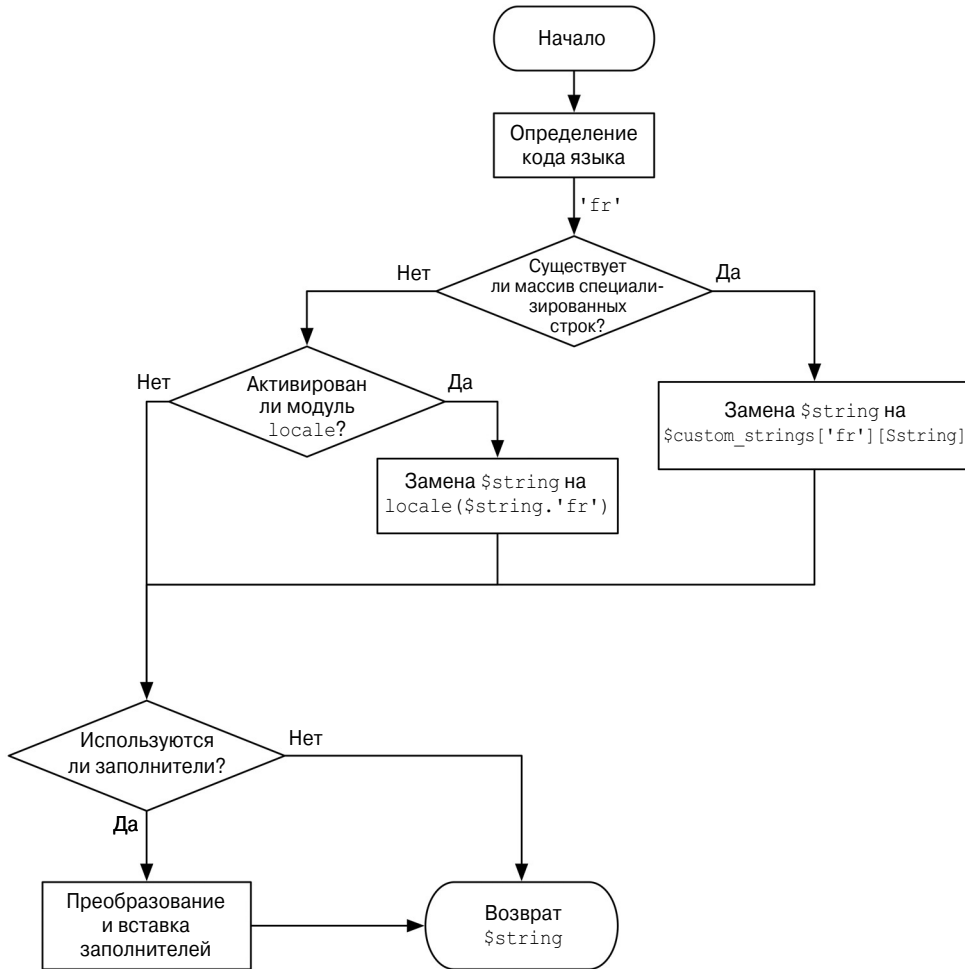


Рис. 19.1. Перевод и вставка текста вместо заполнителей функций `t()` при условии, что в качестве текущего установлен французский язык

Переопределения строк в `settings.php`

Найдите файл `settings.php` (обычно в `sites/default/settings.php`). Возможно, перед внесением изменений этот файл придется сделать доступным для записи, поскольку в целях безопасности Drupal предпочитает защищать его от записи, делая доступным только для чтения. Перейдите в конец файла `settings.php`. Удалите ведущие знаки комментария `#` и добавьте `'home' => 'Sweet Home'` в список значений, предназначенных для перевода.

```

/**
 * Переопределения строк:
 *
 * Для переопределения конкретных строк на сайте с активированным
 * или отключенным модулем locale добавьте запись в этот список.
 * Данная функциональность позволяет изменить небольшое количество
 * строк англоязычного пользовательского интерфейса по умолчанию.

```

```

*
* Для активации удалите ведущие знаки #.
*/
$config['locale_custom_strings_en'] = array(
  'forum' => 'Discussion board',
  '@count min' => '@count minutes',
  'home' => 'Sweet Home',
);

```

Если вы посетите сайт, то заметите, что в навигационной цепочке элемент Home превратился в Sweet Home, как показано на рис. 19.2.

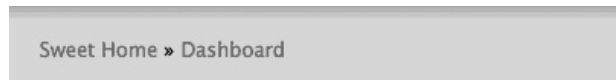


Рис. 19.2. В навигационной цепочке строка Home заменена Sweet Home

Теперь, зная, как переопределяются строки, продолжим и заменим слово Blog словом Journal:

```

$config['locale_custom_strings_en'] = array(
  'Blog' => 'Journal',
);

```

Теперь активируйте модуль blog на странице Modules (Модули). Перейдите к Add content⇒Blog entry (Добавить контент⇒Запись блога). Экран должен выглядеть подобно показанному на рис. 19.3.

 A screenshot of a web form titled "Sweet Home » Add new content" and "Create Blog entry". The form has a "Title *" field with a text input box. Below it is a "Body (Edit summary)" field with a large text area. The form is styled with a light gray header and a white body.

Рис. 19.3. Строка Blog entry не превратилась в Journal entry

В чем же дело? Почему наш специальный массив замен строк был проигнорирован? Дело в том, что строка Blog entry отличается от строки Blog. Нельзя просто указать подстроки для замены; необходимо выполнять сопоставление полной строки.

Как же найти все строки, содержащие слово Blog, чтобы каждое вхождение можно было заменить соответствующим значением Journal? В этом нам поможет модуль locale.

Совет. Использование переопределений строк в файле `settings.php` обеспечивает высокую производительность (но только для небольших наборов строк), потому что не требуется никаких обращений к базе данных; заменяющая строка просто ищется в массиве. При этом, чтобы переопределение строк функционировало, даже не нужно активировать модуль `locale`. Ознакомьтесь также с модулем переопределения строк по адресу <http://drupal.org/project/stringoverrides>.

Замена строк с помощью модуля локализации

Вместо того, чтобы заменять строки, определяя список нестандартных замен строк в файле `settings.php`, для нахождения строк, подлежащих замене, и определения того, какими должны быть эти замены, можно использовать модуль локализации. Применительно к Drupal перевод с одного языка на другой — это набор нестандартных замен строк. Как было упомянуто ранее, когда среда Drupal готовит строку к отображению, она обрабатывает ее функцией `t()`. При обнаружении замены в переводе на текущий язык система применяет эту замену. В противном случае она просто использует исходную строку. Этот процесс, выполняемый функцией `locale()`, в упрощенном виде показан на рис. 19.4. Подход заключается в создании языка с языковым кодом `en-US`, содержащего только те строки, которые требуется заменять.

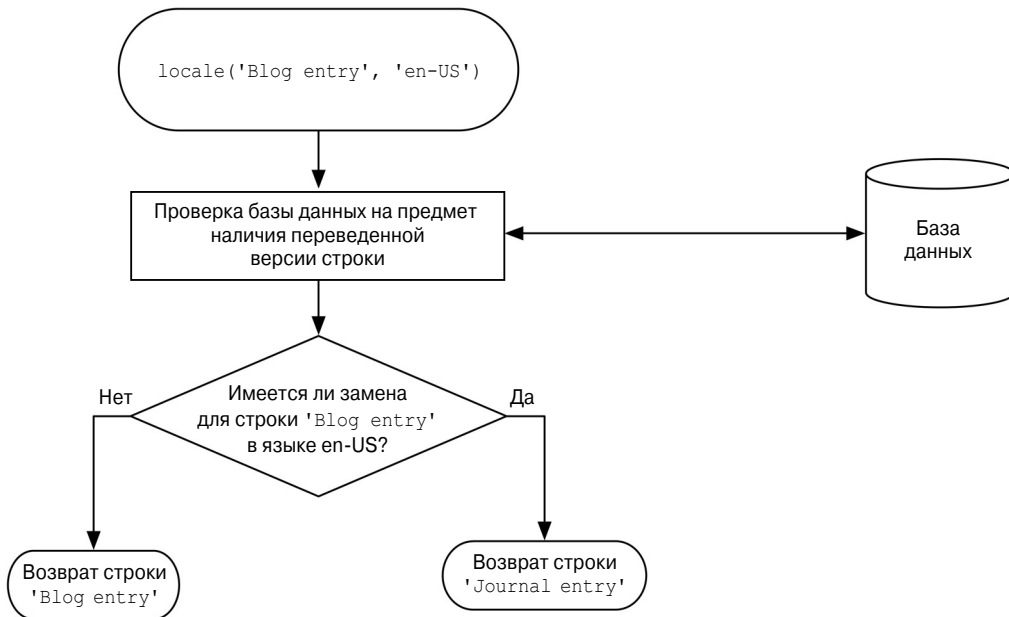


Рис. 19.4. Если модуль локализации не находит строки замены в переводе для текущего языка, он возвращается к использованию исходной строки

Давайте начнем процесс замены любой строки, содержащей "blog", строкой, содержащей "journal". Поскольку при отсутствии найденного перевода Drupal вернется к использованию исходной строки, системе нужно предоставить только те строки, которые требуется изменить. Строки можно поместить в специальный язык и позволить Drupal возвращаться к исходным строкам при отсутствии соответствующего перевода. Сначала добавим специальный язык, который будет содержать специальные строки.

Соответствующий интерфейс показан на рис. 19.5. Назовем язык `English-custom` и установим `en-US` в качестве кода языка и префикса пути. Теперь необходимо перейти к странице `Configuration` ⇒ `Languages` ⇒ `Add a Language` (Конфигурация ⇒ Языки ⇒ Добавить язык).

Sweet Home » Dashboard » Configuration » Regional and language » Languages

Languages o LIST DETECTION AND SELECTION

Add a language to be supported by your site. If your desired language is not available in the Language name drop-down, click Custom language and provide a language code and other details manually. When providing a language code manually, be sure to enter a standardized language code, since this code may be used by browsers to determine an appropriate display language.

▼ PREDEFINED LANGUAGE

Language name
Afrikaans (afrikaans) ▼

Use the Custom language section below if your desired language does not appear in this list.

Add language

▼ CUSTOM LANGUAGE

Language code *
en-US
RFC 4646 compliant language identifier. Language codes typically use a country code, and optionally, a script or regional variant name. Examples: "en", "en-US" and "zh-Hant".

Language name in English *
English-Custom
Name of the language in English. Will be available for translation in all languages.

Native language name *
English-custom
Name of the language in the language being added.

Path prefix language code
en-US
Language code or other custom text to use as a path prefix for URL language detection, if your Detection and selection settings use URL path prefixes. For the default language, this value may be left blank. Modifying this value may break existing URLs. Use with caution in a production environment. Example: Specifying "deutsch" as the path prefix code for German results in URLs like "http://example.de/contact" and "http://de.example.com/contact", respectively.

Language domain

URL including protocol to use for this language, if your Detection and selection settings use URL domains. For the default language, this value may be left blank. Modifying this value may break existing URLs. Use with caution in a production environment. Example: Specifying "http://example.de" or "http://de.example.com" as language domains for German results in URLs like "http://example.de/contact" and "http://de.example.com/contact", respectively.

Direction *
☒ Left to right
☐ Right to left
Direction that text in this language is presented.

Add custom language

Рис. 19.5. Добавление специального языка для перевода конкретных строк

Активируйте новый язык и сделайте его языком по умолчанию, как показано на рис. 19.6. Щелкните на кнопке `Save configuration` (Сохранить конфигурацию), снимите отметку с флажка `Enabled` (Включить) рядом с `English` (Английский) и щелкните на кнопке `Save configuration` еще раз (рис. 19.7). Когда включен только один язык, пользователям не придется сталкиваться с вносящим определенную путаницу выбором `Language settings` (Настройки языка) при редактировании своих учетных записей (рис. 19.8).

Итак, мы получили включенный перевод для единственного языка `English-custom`. В настоящее время он пуст, поскольку никаких замен строк пока еще не добавлено. Поэтому для каждой строки `Drupal` выполнит процесс, показанный на рис. 19.4, не сможет найти замену строки для языка `English-custom` и вернется к исходной строке из английского языка. Теперь установим замены некоторых строк. Перейдите к странице `Configuration` ⇒ `Translate interface` (Конфигурация ⇒ Перевести интерфейс), показанной на рис. 19.9.

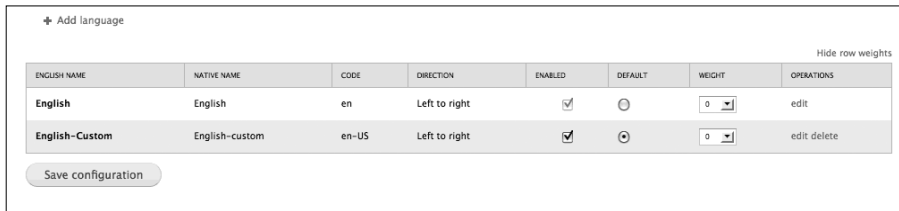


Рис. 19.6. Включение нового языка и его выбор в качестве языка по умолчанию

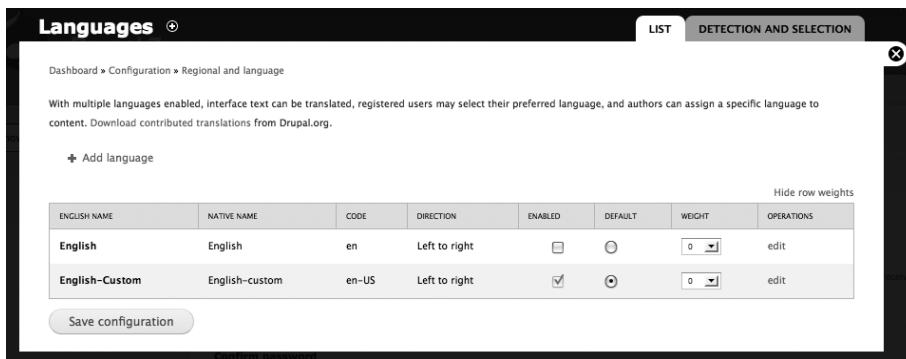


Рис. 19.7. Отключение языка English, чтобы язык English-custom был единственным включенным языком

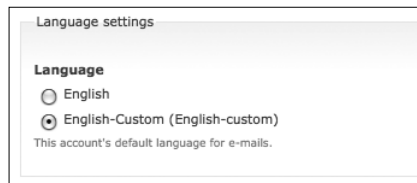


Рис. 19.8. Интерфейс пользователя на странице My account (Моя учетная запись), где пользователь может выбрать язык для сообщений электронной почты, отправляемых сайтом. (Этот интерфейс отображается только при наличии нескольких включенных языков.)

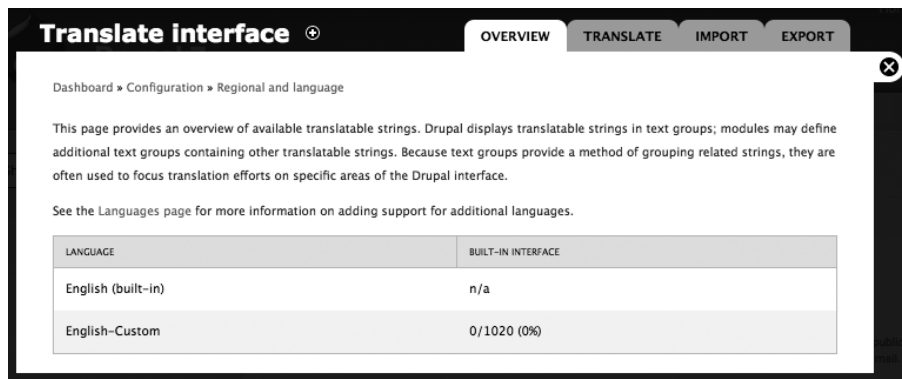


Рис. 19.9. Вкладка Overview (Обзор) страницы Translate interface

Среда Drupal использует оперативный (just-in-time) перевод. При загрузке страницы каждая строка, обрабатываемая функцией `t()`, а равно и функцией `locale()`, если она отсутствует в таблицах `locales_source` и `locales_target` базы данных, добавляется в эти таблицы. Поэтому значения в столбце **Built-in interface** (Встроенный интерфейс) на рис. 19.9 показывают, что функция `t()` обработала 1020 строк, и все они доступны для перевода. Обратитесь к нескольким другим страницам Drupal, после чего вернитесь к этой. Количество строк должно увеличиться, поскольку Drupal встречает все больше и больше элементов интерфейса, которые нуждаются в переводе. Теперь воспользуемся веб-интерфейсом модуля локализации для перевода некоторых строк.

После щелчка на вкладке **Translate** (Перевести) откроется интерфейс поиска, который позволяет найти строки для перевода. Выполним поиск всех этих 1020 или более строк, которые доступны на данный момент. Интерфейс поиска показан на рис. 19.10.

▼ FILTER TRANSLATABLE STRINGS

String contains

Leave blank to show all strings. The search is case sensitive.

Language Search in Limit search to

All languages Both translated and untranslated st All text groups Filter

Рис. 19.10. Интерфейс поиска для отображения строк, которые могут быть переведены

Выбор нашего языка (**English-custom**) и поиск всех строк при пустом поле поиска приведет к отображению всех доступных для перевода строк. Рядом с каждой строкой отображается ссылка **edit** (редактировать). Под списком строк в нижней части страницы снова отображается интерфейс поиска. Поскольку список строк довольно длинный, ограничим его строками, которые содержат слово **“Translate”**. Введите слово **Translate** в поле **String contains** (Строка содержит) и щелкните на кнопке **Filter** (Фильтр). В результате должен отобразиться список, содержащий слово **“Translate”**, как показано на рис. 19.11.

Translate interface

Dashboard » Configuration » Regional and language » Translate interface

This page allows a translator to search for specific translated and untranslated strings, and is used when creating or editing translations. (Note: For translation tasks involving many strings, it may be more convenient to export strings for offline editing in a desktop Gettext translation editor.) Searches may be limited to strings found within a specific text group or in a specific language.

▼ FILTER TRANSLATABLE STRINGS

String contains

Translate

Leave blank to show all strings. The search is case sensitive.

Language Search in Limit search to

All languages Both translated and untranslated st All text groups Filter Reset

TEXT GROUP	STRING	CONTEXT	LANGUAGES	OPERATIONS
Built-in interface	Translate the built in interface and optionally other text. /admin/config/regional/language		en-US	edit delete
Built-in interface	Translate interface texts /admin/people		en-US	edit delete
Built-in interface	Translate interface /admin/config/regional/translate/translate?render=overlay		en-US	edit delete
Built-in interface	Translate /admin/config/regional/translate/translate?render=overlay		en-US	edit delete

Рис. 19.11. Список доступных для перевода строк, содержащих слово **“translate”**, и их состояние

Изменим строку `Translate interface` на `Translate language interface`, щелкнув на ссылке `edit` для этой строки.

После редактирования строки мы возвращаемся на вкладку `Translate` (рис. 19.12). Заголовок страницы должен измениться с `Translate interface` на `Translate language interface`.



Рис. 19.12. Теперь строка `Translate interface` заменена строкой `Translate language interface`

Продолжим и снова выполним поиск строки `Translate`. Теперь в результирующем списке строк зачеркивание должно быть удалено из столбца `Languages` (Языки) для записи `Translate interface`; это указывает на то, что строка была переведена (рис. 19.13).

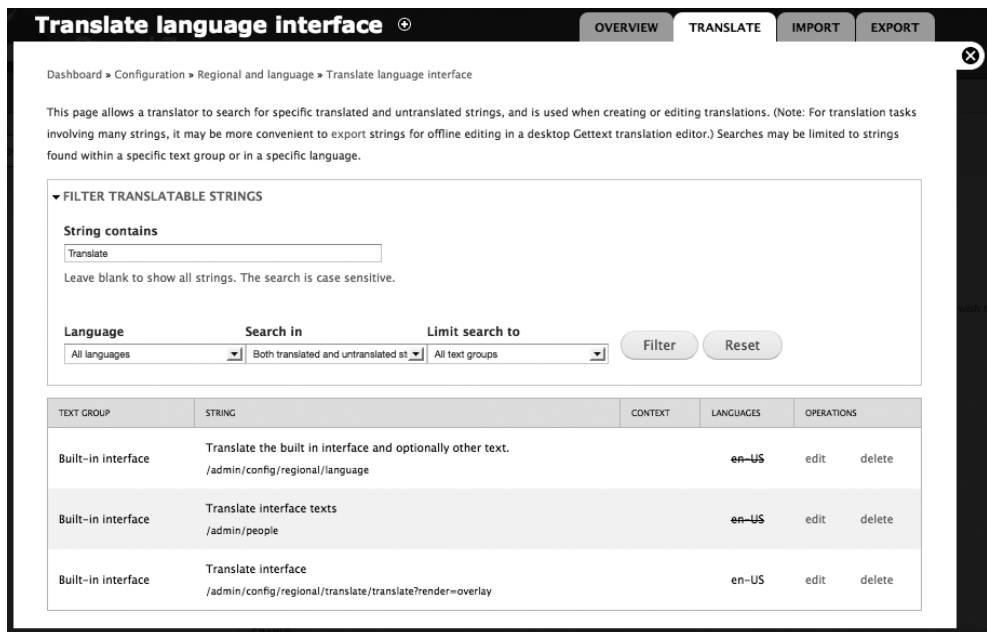


Рис. 19.13. Список доступных для перевода строк после редактирования записи `Translate interface`

Обратите внимание, что отображается исходная строка, а не перевод. Если мы вернемся к вкладке `Overview`, то увидим, что язык `English-custom` содержит одну доступную строку замены.

Теперь, когда мы научились изменять строки, можно приступить к изменению всех слов `blog` на `journal`. После активации модуля `blog` и посещения относящихся к блогам страниц (например, `/node/add/blog` и `blog/1`) соответствующие строки должны быть доступными для перевода. Поиск на странице `Configuration > Translate` чувствителен к регистру символов, поэтому две процедуры поиска по ключевым словам `blog` и `Blog` приведут к отображению всех совпадений и позволят изменить их соответственно на `journal` и `Journal`.

Внимание! Используемый нами метод пригоден для сайтов Drupal и ориентирован на замену определенных строк элементов интерфейса, однако он не полон. Например, если модуль, содержащий слово “blog”, не активирован, перевод этих строк будет пропущен. Более полный метод описан в разделе “Начало нового перевода” далее в главе.

Эти изменения хороши, но несколько раздражает, что URL-адрес для создания новой записи в журнале остается прежним, `http://example.com/?q=node/add/blog`; не должен ли он измениться на `http://example.com/?q=node/add/journal`? Разумеется, должен. Этого можно добиться, активировав модуль `path` и добавив псевдоним `node/add/journal` для существующего системного пути `node/add/blog`. Как видите, все ссылки на `blog` исчезли и сайтом можно пользоваться, не испытывая дискомфорт от отсутствия слова “blog”.

Совет. Облегчает перевод строк модуль от независимых разработчиков `Localization client` (Клиент локализации), доступный по адресу `http://drupal.org/project/l10n_client`. Этот модуль предоставляет страничный интерфейс редактора локализации и интенсивно использует AJAX.

Экспорт перевода

После выполнения действий по выбору и переводу строк, которые требовалось изменить, было неэффективно проделывать все это снова при установке следующего сайта Drupal. Используя вкладку `Export` (Экспорт) на странице `Configuration` ⇒ `Translate`, можно сохранить перевод в специальном файле, который называется файлом переносимых объектов (`.po`). Этот файл будет содержать все строки, передаваемые Drupal посредством функции `t()`, равно как и любые строки замены, определенные нами.

Файлы переносимых объектов

Ниже приведено несколько начальных строк файла, которые получены в результате экспорта перевода `English-custom`:

```
# English-Custom translation of Drupal 7
# Generated by admin <toddtomlinson@serverlogic.com>
#
msgid ""
msgstr ""
"Project-Id-Version: PROJECT VERSION\n"
"POT-Creation-Date: 2010-08-08 06:01-0700\n"
"PO-Revision-Date: 2010-08-08 06:01-0700\n"
"Last-Translator: NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <EMAIL@ADDRESS>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: 8bit\n"

#: misc/drupal.js
msgid "An AJAX HTTP error occurred."
msgstr ""

#: misc/drupal.js
msgid "HTTP Result Code: !status"
msgstr ""

...
```

Файл `.po` состоит из нескольких заголовков метаданных, сопровождаемых переведенными строками. Каждая строка содержит три компонента: комментарий, который показывает, где строка появилась в первый раз, `msgid`, обозначающий исходную строку, и `msgstr`, который обозначает переведенную строку, предназначенную для использования. Полное описание формата `.po` можно найти по адресу www.gnu.org/software/gettext/manual/gettext.htmlgettext.html#PO-Files.

Теперь файл `en-US.po` может быть импортирован на другой сайт Drupal (с активированным модулем `locale`) с помощью вкладки **Import** (Импорт) интерфейса **Configuration** ⇒ **Translate**.

Шаблоны переносимых объектов

В то время как перевод состоит из некоторых метаданных и множества исходных и переведенных строк, файл шаблонов переносимых объектов (`.pot`) содержит все доступные для перевода строки, но без переведенных строк. Это полезно при переводе с нуля или если требуется определить, появились ли какие-то новые строки в последней версии Drupal, прежде чем изменять сайт (другой способ выяснить это состоит в модернизации своего Drupal-сайта и поиске непереведенных строк, как было описано в разделе “Замена встроенных строк специальными строками”).

Начало нового перевода

Файлы перевода для ядра Drupal, а также для многих дополнительных модулей на десятках языков можно загрузить из сайта <http://localize.drupal.org>. В этом разделе Drupal.org вы найдете файлы перевода для конкретного языка (файлы `.po`), которые могут быть загружены и установлены, немедленно предоставляя многоязычные возможности на вашем сайте. Можно также установить модуль **Localized Drupal** (Локализованный Drupal; http://drupal.org/project/l10_install), который программно извлекает переводы интерфейса пользователя из <http://localize.drupal.org>, и модуль **Localization Client** (Клиент локализации; http://drupal.org/l10n_client), который предоставляет простые способы локализации интерфейса посредством интерфейса страничного редактора локализации — обеспечивая возможность настройки перевода интерфейса непосредственно на просматриваемых веб-страницах.

Создание файлов `.pot` с помощью модуля извлечения шаблонов перевода

Дополнительный модуль извлечения шаблонов перевода (<http://drupal.org/project/potx>), может автоматически генерировать файлы `.pot`. Это полезно, если вы написали собственный модуль или загрузили дополнительный, для которого перевод пока не существует. Модуль извлечения шаблонов перевода содержит как версию командной строки, так и версию с веб-интерфейсом. Если вы знакомы с программой `xgettext` для UNIX, то можете рассматривать этот модуль как Drupal-версию упомянутой программы. При генерировании файлов перевода этот модуль пользуется поддержкой <http://localize.drupal.org>.

Создание файла `.pot` для модуля

Давайте сгенерируем файл `.pot` для модуля объявления о вакансиях (`job_post`), который был создан в главе 8.

Вначале необходимо установить модуль извлечения шаблонов перевода. Скопируйте ссылку загрузки из <http://drupal.org/project/potx> и установите модуль, перейдя к странице **Modules** (Модули) и щелкнув на ссылке **Install new module** (Установить новый модуль). Вставьте ссылку загрузки в текстовое поле **Install from a URL** (Установить из URL) и щелкните на кнопке **Install** (Установить).

Использование командной строки

Скопируйте файлы `potx.inc` и `potx-cli.php` из каталога модуля `potx` в каталог модуля `job_post` в каталоге `sites/all/modules/custom/job_post`. Затем потребуется запустить модуль извлечения, чтобы он мог создать файлы `.pot`.

Внимание! К сайту Drupal добавляется исполняемый сценарий PHP, которому нужны права доступа к каталогу, где он выполняется (чтобы он мог записать файл `.pot`). Всегда извлекайте шаблон в копии сайта на своем компьютере в разработке, а не на действующем сайте.

Ниже показаны результаты работы модуля извлечения:

```
$ cd sites/all/modules/custom/job_post
$ php potx-cli.php
Processing sponsor.tpl.php...
Processing job_post.module...
Processing job_post.install...
Processing job_post.info...
```

Посмотрим, что было сгенерировано:

<code>general.pot</code>	<code>job_post.install</code>	<code>potx.inc</code>
<code>installer.pot</code>	<code>job_post.module</code>	<code>sponsor.tpl.php</code>
<code>job_post.info</code>	<code>potx-cli.php</code>	

В результате запуска сценария модуля извлечения появляется новый файл `general.pot`, содержащий строки из файлов `sponsor.tpl.php`, `job_post.module`, `job_post.info` и `job_post.install`. По умолчанию этот сценарий помещает все строки в файл `general.pot`, однако при необходимости может сгенерировать и отдельные файлы. Чтобы ознакомиться с различными опциями, предлагаемыми сценарием модуля извлечения, введите следующую команду:

```
$ php potx-cli.php --help
```

В данном случае удобно сосредоточить все строки в одном файле. Если бы этот шаблон перевода требовалось использовать совместно с другими разработчиками, нужно было создать подкаталог `translations` внутри каталога `annotate`, переместить `general.pot` в каталог `translations` и переименовать его в `annotate.pot`. Если затем выполнять перевод, скажем, на французский язык, открыв комбинированный файл `.pot`, переведя строки и сохранив его как `fr.po`, каталог модуля аннотирования должен был бы выглядеть следующим образом:

<code>general.pot</code>	<code>job_post.install</code>	<code>potx.inc</code>
<code>installer.pot</code>	<code>job_post.module</code>	<code>sponsor.tpl.php</code>
<code>job_post.info</code>	<code>potx-cli.php</code>	
<code>translations/ annotate.pot</code>		
<code>fr.po</code>		

Использование модуля извлечения с веб-интерфейсом

Вместо применения командной строки, строки из модуля можно извлекать также и с помощью веб-интерфейса пользователя, предоставляемого модулем извлечения шаблонов перевода. Удостоверившись, что модуль установлен, перейдите к странице **Modules** и включите как модуль `job_post`, так и модуль извлечения шаблонов перевода. Затем перейдите к странице **Configuration** ⇒ **Translate interface** и обратите внимание на новую вкладку **Extract** (Извлечение). Достаточно щелкнуть на ней, чтобы получить воз-

возможность генерации файла `.pot`, разворачивая каталог `sites/all/modules`, а в этом расширенном списке выбирая модуль, для которого требуется сгенерировать перевод. Затем выберите переключатель `Language independent template` (Шаблон, не зависящий от языка) и щелкните на кнопке `Extract` (Извлечь), как показано на рис. 19.14. Файл `.pot` будет загружен через браузер. Затем файл `.pot` можно поместить в каталог модуля, как это было сделано в случае с модулем извлечения командной строки.

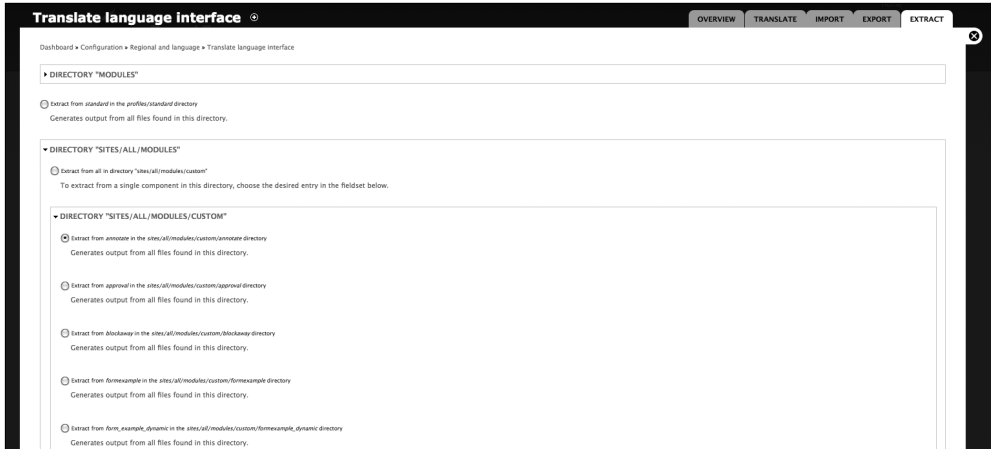


Рис. 19.14. Извлечение файла `.pot` с помощью веб-интерфейса пользователя модуля извлечения шаблонов перевода

Создание файлов `.pot` для всего сайта

Если необходимо создать файлы `.pot` для всех переводимых строк сайта, поместите файлы `potx.inc` и `potx-cli.php` в корень сайта, удостоверьтесь в наличии права на запись в текущем каталоге и запустите `potx-cli.php`. Если требуется генерировать файлы `.pot` с той же структурой, что и доступные в <http://drupal.org/project/Translations>, нужно запустить сценарий из командной строки с параметром `mode`, установленным в `core`:

```
$ php potx-cli.php --mode=core
```

Сценарий всегда выводит файлы `.pot` в тот же каталог, откуда он запущен; например, `modules-aggregator.pot` будет создан в корневом каталоге сайта, а не в `modules/aggregator/`. Имя файла `.pot` отражает его местоположение. Поэтому в предыдущем примере будет сгенерирован файл `sites-all-modules-custom-annotate.pot`.

Установка перевода на определенный язык

Среда Drupal может быть инсталлирована на языке, отличном от английского. Перевод на определенный язык может также быть добавлен позднее. Рассмотрим обе возможности.

Установка перевода во время инсталляции

Инсталлятор Drupal распознает переводы инсталлятора с помощью функции `st()`, а не `t()`, которая недоступна инсталлятору во время выполнения, поскольку среда Drupal на этот момент еще не установлена. Переводы инсталлятора предлагаются в качестве вариантов во время инсталляции и основаны на файле `installer.pot` (см. раздел «Начало нового перевода» ранее в главе).

Чтобы увидеть возможности перевода инсталлятора в действии, загрузим французский перевод Drupal из www.drupal.org/project/translations. Мы получим файл `fr-7.x-1.1.tar.gz`. Расширение `.tar.gz` свидетельствует, что это — файл `.tar`, который был упакован с использованием алгоритма сжатия GZIP. Один из способов извлечения содержимого этого файла — применение Unix-утилиты `tar`:

```
$ tar -xzf fr-7.x-1.1.tar.gz
```

Внимание! Этот файл содержит структуру каталогов, которая отражает структуру каталогов Drupal. При ее извлечении соблюдайте осторожность, используя метод, который объединяет извлекаемую структуру с существующей структурой каталогов Drupal. Модуль извлечения по умолчанию в Mac OS X делает это некорректно. Если после извлечения вы получите каталог `fr-6.x-1.x-dev`, значит, объединение каталогов не произошло. Снимки экрана, демонстрирующие правильный способ извлечения, можно найти по адресу www.lullabot.com/videocast/installing-drupal-translation.

После успешного извлечения перевода в каталогах Drupal должны появиться дополнительные папки `translations`. Например, папка `profiles/default` (содержащая профиль инсталляции Drupal по умолчанию) теперь содержит подпапку `translations`, в которой находится файл `fr.po`. Это — французский перевод инсталляции. При запуске инсталлятора Drupal отобразится новый представленный вариант, как показано на рис. 19.15.

Если вы выберете French (Французский), инсталляция продолжится на французском языке, и языком по умолчанию для сайта будет французский.



Рис. 19.15. Когда файл `.po` присутствует в каталоге профиля инсталляции, инсталлятор Drupal позволяет выбрать язык для него

Инсталляция перевода на существующем сайте

Чтобы установить язык перевода на существующем сайте, необходимо добавить язык, перейдя на страницу Languages (Языки) и щелкнув на вкладке Add language (Добавить язык). Затем понадобится выбрать язык и щелкнуть на кнопке Add language (Добавить язык), как показано на рис. 19.16. Новый язык отобразится в таблице на странице Configuration⇒Languages.

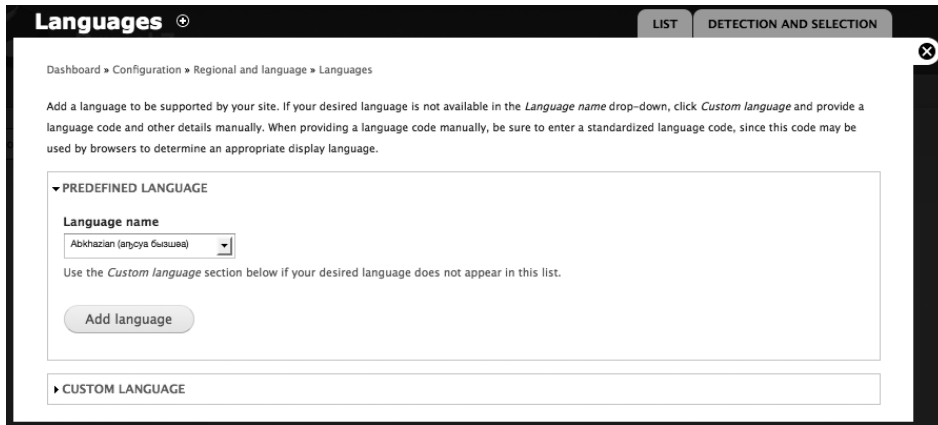


Рис. 19.16. Установка языка

Поддержка языков с письмом справа налево

Направление языка отображается в списке языковых переводов, которые были добавлены в Drupal, как показано на рис. 19.17.

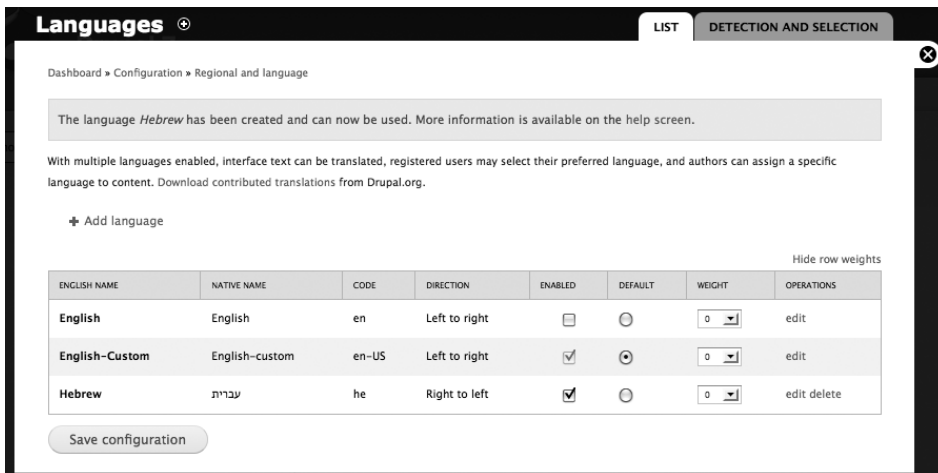


Рис. 19.17. Языки с письмом справа налево можно идентифицировать в столбце Direction (Направление) таблицы языков

Поддержка Drupal языков с письмом справа налево, таких как иврит, осуществляется на уровне тем. Когда система Drupal узнает, что таблица стилей должна быть включена в текущую страницу, а направление письма в текущем языке — справа налево (right-to-left — RTL), она попытается найти имя соответствующей таблицы стилей, оканчивающееся на `-rtl.css`. Если эта таблица стилей существует, она будет загружена в дополнение к запрошенной таблице стилей. Логика процесса продемонстрирована на рис. 19.18. Таким образом, темы, поддерживающие языки с письмом справа налево, обычно имеют стили, определенные в главной таблице стилей, и CSS переопределяет соответствующую таблицу стилей письма “справа налево”.

Например, если текущий язык — иврит, и тема — Seven, то когда Drupal добавляет таблицу стилей `themes/seven/style.css`, файл `themes/seven/style-rtl.css` также включается. Проверьте таблицы стилей “справа налево” в темах Drupal по умолчанию, чтобы увидеть, какие виды CSS-элементов они переопределяют.

Направление письма языка можно изменить на странице Configuration ⇒ Languages щелчком на ссылке `edit` (редактировать) соответствующего языка.

Проверка направления письма текущего языка осуществляется в коде следующим образом:

```
if (defined('LANGUAGE_RTL') && $language->direction == LANGUAGE_RTL) {
  // Выполнение каких-либо действий.
}
```

Это возможно благодаря тому, что константа `LANGUAGE_RTL` определена модулем локализации, поэтому если модуль локализации не загружен, поддержка языков с письмом справа налево недоступна.

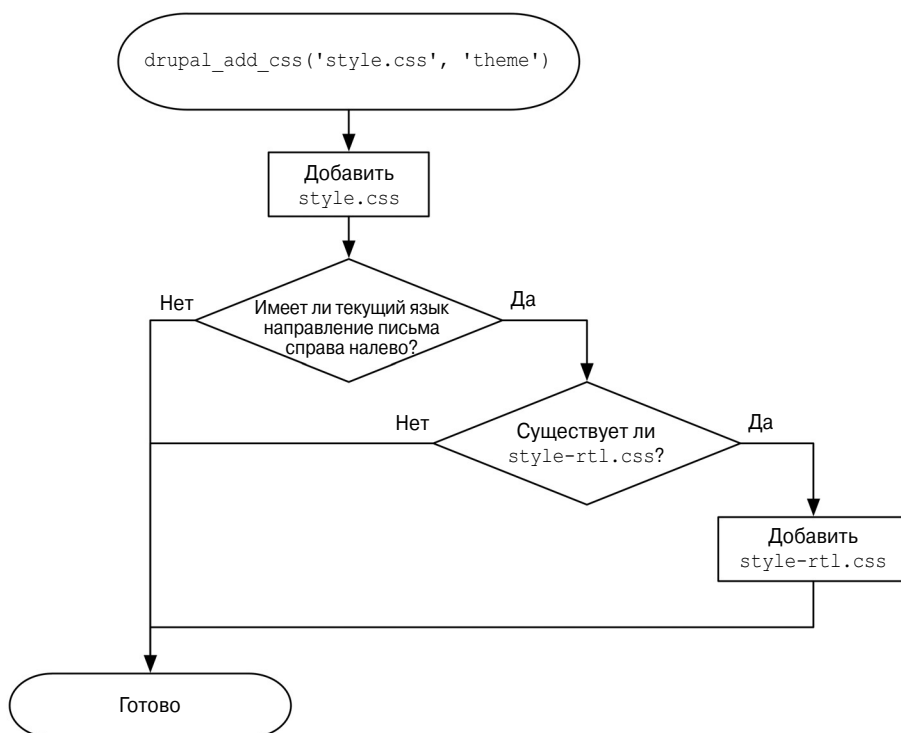


Рис. 19.18. Если текущий язык относится к языкам с письмом справа налево, будет включена дополнительная таблица стилей, если она существует

Согласование языков

В Drupal реализовано большинство распространенных способов определения языка пользователя, поэтому когда на сайте Drupal включено несколько языков, применяется язык, предпочитаемый пользователем. В следующих разделах мы будем исходить из того, что установлен французский перевод Drupal, как было описано в предыдущем разделе. Способ определения Drupal языковых настроек конфигурируется на вклад-

ке Detection and Selection (Обнаружение и выбор) экрана Configuration ⇒ Languages. Соответствующий интерфейс пользователя показан на рис. 19.19. Рассмотрим каждую из этих опций.

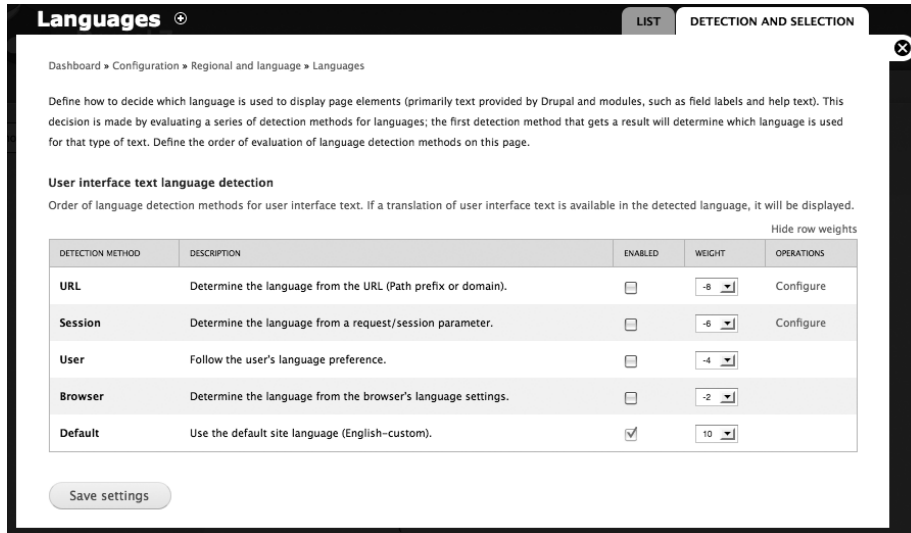


Рис. 19.19. Возможные настройки согласования языков

Опция Default (По умолчанию)

Эта опция является простейшей и выбирается по умолчанию. Язык, устанавливаемый в качестве языка по умолчанию, применяется для всех пользователей при отображении страниц. На рис. 19.17 показан интерфейс пользователя, в котором указан язык по умолчанию.

Опция User (Пользовательский)

Если включено более одного языка, пользователи на своих страницах My account (Моя учетная запись) увидят набор полей, показанный на рис. 19.20.

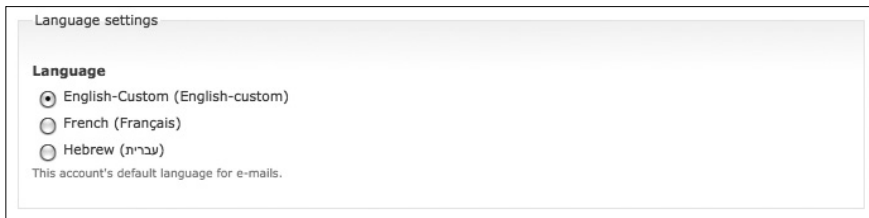


Рис. 19.20. Выбор специфичного для пользователя языка сообщений электронной почты

Выбранный пользователем язык можно извлечь следующим образом:

```
// Извлечь язык, предпочитаемый пользователем с идентификатором 3.
$account = user_load(1); //В примере применяется учетная запись администратора
$language = user_preferred_language($account);
```

Если пользователь не установил предпочитаемый язык, будет возвращен язык по умолчанию для сайта. В результате получается объект `language` (подробнее этот объект описан в следующем разделе). Когда в группе `Language negotiation` (Согласование языков) выбран переключатель `None` (Отсутствует), предпочитаемый пользователем язык используется *только* для определения языка сообщений электронной почты, отправляемых с сайта. Когда настройка `Language negotiation` установлена в `None`, предпочитаемый пользователем язык не оказывает эффекта на язык, используемый для отображения страниц.

Глобальный объект `$language`

Текущий язык можно определить программно, проверив переменную `$language`, являющуюся объектом. Переменная инициализируется в течение фазы `DRUPAL_BOOTSTRAP_LANGUAGE` загрузки. Посмотреть, как выглядит объект, можно с помощью функции `var_dump()`:

```
global $language;
var_dump($language);
```

Результат показан ниже:

```
object(stdClass) (11) {
  ["language"] => string(2) "fr"
  ["name"] => string(6) "French"
  ["native"] => string(9) "Francais"
  ["direction"] => string(1) "0"
  ["enabled"] => int(1)
  ["plurals"] => string(1) "2"
  ["formula"] => string(6) "($n>1)"
  ["domain"] => string(0) ""
  ["prefix"] => string(2) "fr"
  ["weight"] => string(1) "0"
  ["javascript"]=> string(0) ""
}
```

Идентификатор языка (такой как `fr` в предыдущем примере) в соответствии с документом RFC 4646 может быть извлечен посредством свойства `language` объекта `$language`:

```
global $language;
$lang = $language->language;
```

Опция `Path prefix only` (Только префикс пути)

Когда согласование языков установлено в `Path Prefix Only` (Только префикс пути), существует только две возможности. Либо префикс пути языка находится в пути, либо используется язык по умолчанию. Например, предположим, что вы создаете сайт, поддерживающий как англоязычных, так и франкоязычных пользователей. Английский — язык сайта по умолчанию, но французский перевод также установлен и включен. Переход к странице `Configuration⇒Languages` и щелчок на ссылке `edit` рядом с языком `French` приведет к отображению интерфейса пользователя, показанного на рис. 19.21. Обратите внимание, что в поле `Path prefix language code` (Код языка в префиксе пути) указано `fr`. Это значение может быть заменено любой строкой.

Когда префикс пути установлен в `fr`, система `Drupal` определяет текущий язык, просматривая запрошенный URL. Этот процесс показан на рис. 19.22.

Edit language ⓘ

Language code
fr

Language name in English *
French
Name of the language in English. Will be available for translation in all languages.

Native language name *
Français
Name of the language in the language being added.

Path prefix language code
fr
Language code or other custom text to use as a path prefix for URL language detection, if your *Detection and selection* settings use URL path prefixes. For the default language, this value may be left blank. **Modifying this value may break existing URLs. Use with caution in a production environment.** Example: Specifying "deutsch" as the path prefix code for German results in URLs like "example.com/deutsch/contact".

Language domain

URL including protocol to use for this language, if your *Detection and selection* settings use URL domains. For the default language, this value may be left blank. **Modifying this value may break existing URLs. Use with caution in a production environment.** Example: Specifying "http://example.de" or "http://de.example.com" as language domains for German results in URLs like "http://example.de/contact" and "http://de.example.com/contact", respectively.

Direction *
☒ Left to right
☐ Right to left
 Direction that text in this language is presented.

Save language

Рис. 19.21. Страница Edit language (Редактирование языка) интерфейса пользователя с полем Path prefix language code

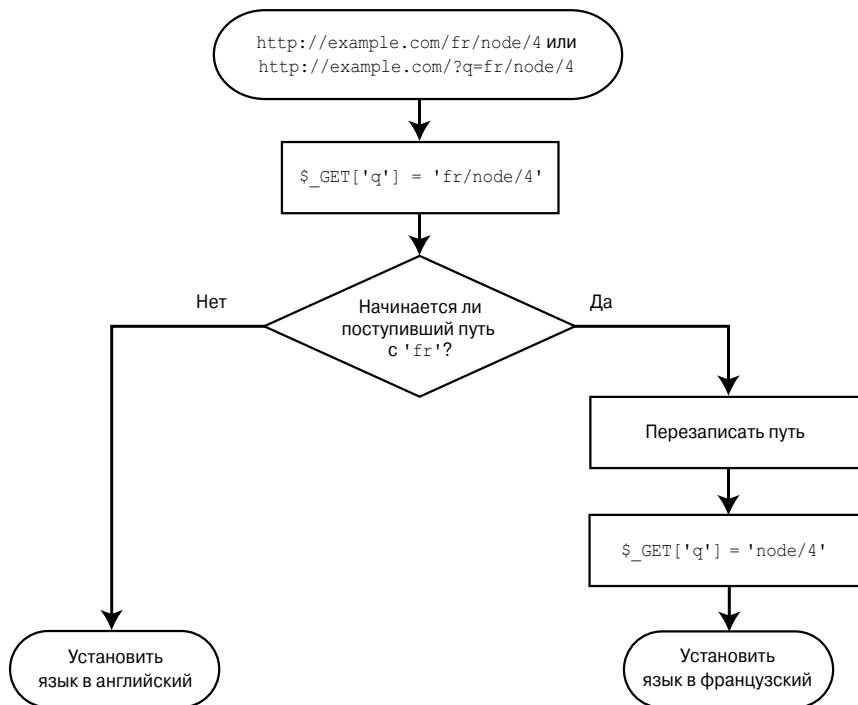


Рис. 19.22. Определение языка с использованием префикса пути `fr`

Опция Path prefix with language fallback (Префикс пути с обходом языка)

Когда согласование языка установлено в Path prefix with language fallback (Префикс пути с обходом языка), среда Drupal сначала просматривает префикс пути. Если соответствие не обнаружено, проверяется предпочитаемый язык пользователя в `$user->language`. Если пользователь не выбрал предпочитаемый язык, Drupal пытается определить предпочитаемый язык пользователя по заголовку `Accept-language` HTTP-запроса браузера. Если и браузер не указывает предпочитаемый язык, используется язык, установленный по умолчанию для сайта. Если предположить, что язык сайта по умолчанию — английский, и включены одновременно французский и иврит, то процесс определения языка выглядит так, как показано на рис. 19.23.

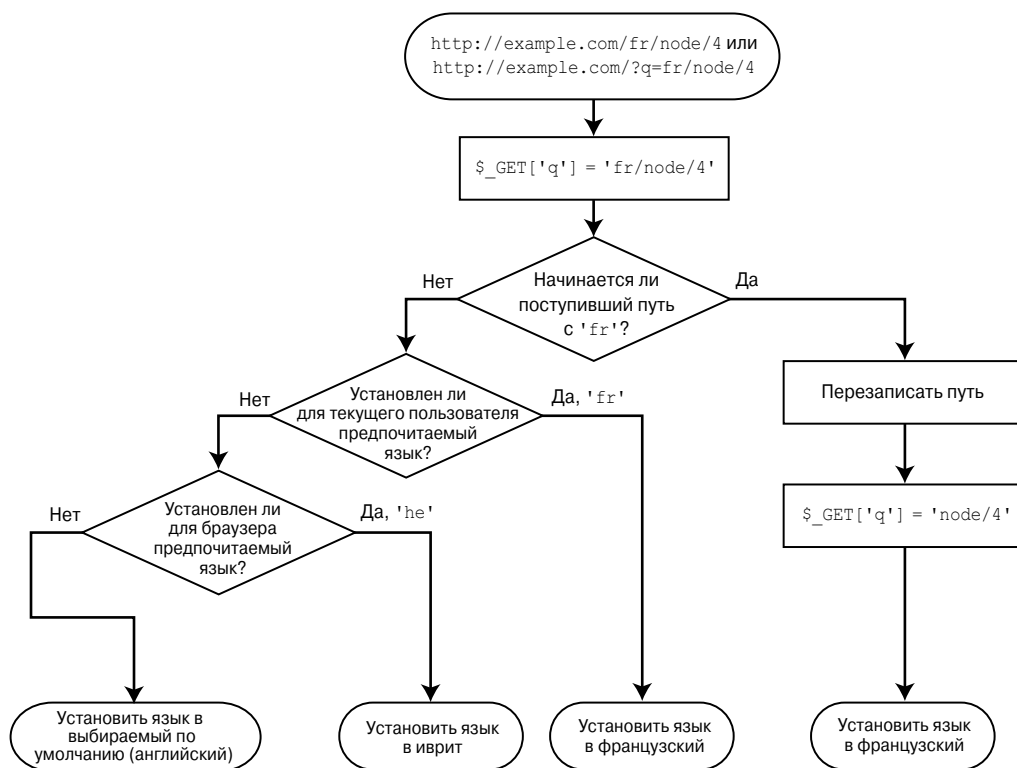


Рис. 19.23. Определение языка при согласовании Path prefix with language fallback

Опция URL Only (Только URL)

Когда для согласования языка выбрана эта настройка, среда Drupal будет определять текущий язык, пытаясь сопоставить текущий URL с доменом языка, указанным в поле Language domain (Домен языка) страницы Edit language (Редактирование языка) языка (см. рис. 19.21). Например, если языком по умолчанию является английский, указание `http://fr.example.com` в качестве домена для французского языка установит текущим языком французский для пользователей, посещающих `http://fr.example.com/?q=node/2`, и английский — для пользователей, посещающих `http://example.com/?q=node/2`.

На заметку! Когда настройка Language negotiation установлена в Domain name only (Только имя домена), установка предпочитаемого языка пользователя на странице My account и настройки клиентского браузера игнорируются.

Перевод контента

До сих пор наше внимание было сосредоточено только на переводе интерфейса пользователя Drupal. А что насчет контента сайта? Как только текущая установка языка определена, существует вероятность, что пользователь пожелает просматривать контент сайта на выбранном языке. Посмотрим, как работает перевод контента.

Начальное знакомство с модулем перевода контента

Система Drupal поставляется со встроенной возможностью управления переводом контента, а именно — с модулем перевода контента. Этот модуль добавляет дополнительную многоязыковую поддержку и опции управления переводом к типам контента Drupal.

Многоязыковая поддержка

После посещения страницы Modules (Модули) и включения модулей локализации (locale) и перевода контента (Content translation) в наборе полей Publishing options (Параметры публикации) каждого типа контента появляется опция Multilingual support (Многоязыковая поддержка). Чтобы просмотреть настройки, перейдите к странице Structure⇒Content types (Структура⇒Типы контента) и щелкните на ссылке edit для типа контента Basic page (Базовая страница). Развертывание набора полей Publishing options должно привести к отображению новых настроек Multilingual support, как показано на рис. 19.24.

Щелкните на переключателе Enabled (Включено) и сохраните тип контента. Если вы теперь перейдете к странице Create content ⇒ Page (Создать контент ⇒ Страница), то в форме создания контента увидите новое раскрывающееся поле, которое позволяет выбрать язык для вывода контента либо указать, что контент является нейтральным к языку (Language neutral). Поле показано на рис. 19.25.

The screenshot shows the 'Basic page' content type settings in Drupal. On the left is a sidebar with navigation links: 'Submission form settings', 'Publishing options' (which is highlighted), 'Display settings', 'Comment settings' (with subtext 'Hidden, Threading, 50 comments per page'), and 'Menu settings'. The main content area is titled 'Default options' and contains several checkboxes: 'Published' (checked), 'Promoted to front page', 'Sticky at top of lists', and 'Create new revision'. Below these is a note: 'Users with the *Administer content* permission will be able to override these options.' The 'Multilingual support' section has three radio buttons: 'Disabled', 'Enabled', and 'Enabled, with translation' (which is selected). Below the radio buttons is a detailed explanation: 'Enable multilingual support for this content type. If enabled, a language selection field will be added to the editing form, allowing you to select from one of the enabled languages. You can also turn on translation for this content type, which lets you have content translated to any of the enabled languages. If disabled, new posts are saved with the default language. Existing content will not be affected by changing this option.'

Рис. 19.24. Настройки многоязыковой поддержки для типа контента

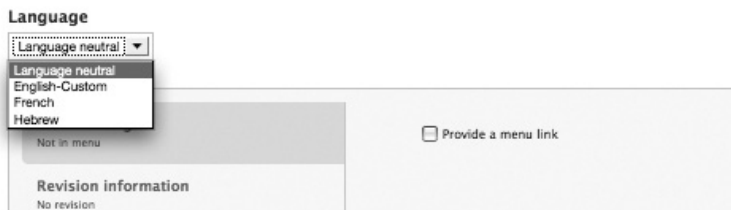


Рис. 19.25. Поле выбора языка в форме создания контента

После создания нескольких страниц на разных языках можно заметить, что административная страница контента Administer⇒Content management⇒Content (Администрирование⇒Управление контентом⇒Контент) изменилась, отражая язык сообщения. Также добавилась опция фильтрации контента по языку, как показано на рис. 19.26.

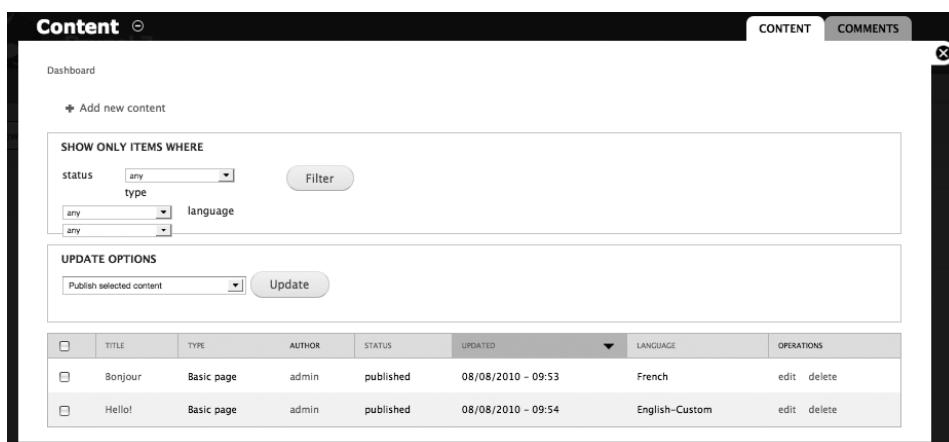


Рис. 19.26. Страница администрирования контента с включенной многоязыковой поддержкой

Многоязыковая поддержка с переводом

Возможность создания контента на нескольких языках — вещь хорошая. Однако большинство сайтов не предоставляют часть контента на английском, а часть — на французском. Вместо этого обычно контент на французском языке представляет собой перевод с английского (или наоборот). Такое становится возможным, когда опция Multilingual support для типа контента установлена в Enabled, with translation (Включено, с переводом) (см. рис. 19.24). При этом применяется следующий подход.

1. Сообщение создается на одном языке. Это — исходное сообщение.
2. Затем создаются переводы этого сообщения.

Рассмотрим эти шаги на примере. Сначала удостоверимся, что текущая настройка Multilingual support для типа контента Basic page установлена в Enabled, with translation. Затем создадим простую страницу на английском языке. Перейдите к странице Create content⇒Page и введите Hello в качестве заголовка и Hello my friends — в качестве тела страницы. Задайте язык English (Английский) и щелкните на кнопке Save (Сохранить). В дополнение к обычным вкладкам View (Вид) и Edit (Редактирование) должна отобразиться вкладка Translate (Перевести); это показано на рис. 19.27.

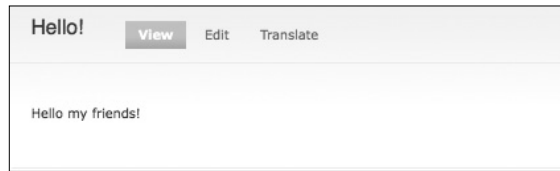


Рис. 19.27. Теперь нота содержит вкладку для выполнения перевода

Щелчок на вкладке Translate приводит к отображению итогов по состоянию перевода сообщения. Как показано на рис. 19.28, исходное сообщение существует только на английском языке. Теперь создадим французский перевод, щелкнув на ссылке add translation (добавить перевод).

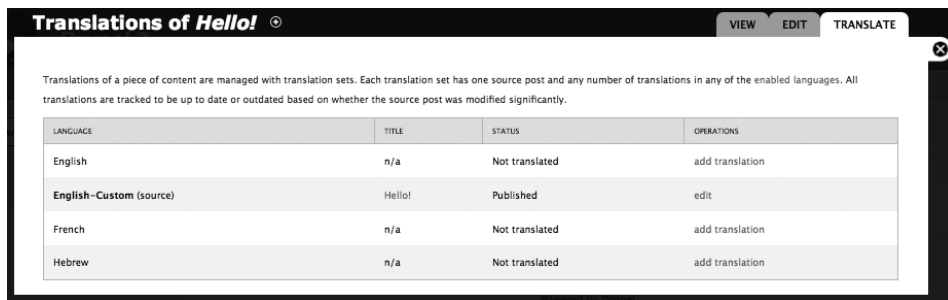


Рис. 19.28. Щелчок на вкладке Translate ведет к отображению итогов по состоянию перевода

Щелчок на ссылке add translation приводит к возврату на страницу редактирования ноды. Но на этот раз в качестве языка установлен French (Французский). Введите *Bonjour* в качестве заголовка и *Ayez un beau jour* — в качестве тела. После щелчка на кнопке Save будет добавлена новая нота. Drupal автоматически создаст ссылки перехода между исходной нодой и переводами, помеченные соответствующим языком. Внешний вид французского перевода исходной ноды, когда исходная нота была на английском языке, а дополнительный перевод — на французском, можно видеть на рис. 19.29.

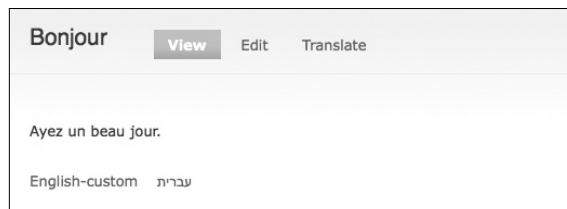


Рис. 19.29. Французский перевод исходной ноды содержит ссылки на английскую версию и версию на иврите

Ссылки строятся путем реализации функции `hook_node_view()` в `modules/translation/translation.module`:

```
/**
 * Реализация hook_node_view().
 *
 * Отобразить ссылки трансляции с родными названиями языков,
 * если нота является частью набора перевода.
 */
```

```
function translation_node_view($node, $view_mode) {
  if (isset($node->tnid) &&
      $translations = translation_node_get_translations($node->tnid))
  {
    $path = 'node/' . $node->nid;
    $links =
      language_negotiation_get_switch_links(LANGUAGE_TYPE_INTERFACE, $path);
    if (is_object($links)) {
      $links = $links->links;

      // Не показывать ссылку на ту же самую ноду.
      unset($links[$node->language]);
      $node->content['links']['translation'] = array(
        '#theme' => 'links__translation_node',
        '#links' => $links,
        '#attributes' => array('class' => array('links', 'inline')),
      );
    }
  }
}
```

В дополнение к сгенерированным ссылкам, модуль локализации предоставляет блок переключателя языков, который может быть активирован на странице Structure⇒Blocks (Структура⇒Блоки). Блок переключателя языка будет отображаться, только если активирована поддержка нескольких языков, а настройка Detection and Selection (Выявление и выбор) отличается от Default. Блок переключения языка показан на рис. 19.30.

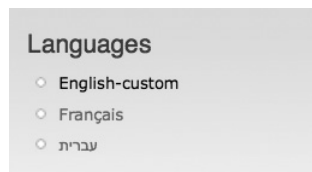


Рис. 19.30. Блок переключения языка

Теперь вернемся к рассмотрению исходных нод и их переводов. Если нода является исходной, то при ее редактировании в форме редактирования ноды отобразится дополнительный набор полей Translation settings (Настройки перевода). Этот набор полей содержит единственный флажок Flag translations as outdated (Пометить переводы как устаревшие), как показано на рис. 19.31.

▼ TRANSLATION SETTINGS

☐ Flag translations as outdated

If you made a significant change, which means translations should be updated, you can flag all translations of this post as outdated. This will not change any other property of those posts, like whether they are published or not.

Рис. 19.31. Набор полей Translation settings в форме редактирования исходной ноды

Упомянутый флажок используется для указания того, что редактирование исходной ноды — достаточно веская причина для повторного перевода. Отметка этого флажка просто вызывает появление слова “outdated” (“устаревший”) при просмотре состояния перевода ноды. Сравните рис. 19.28 и 19.32.

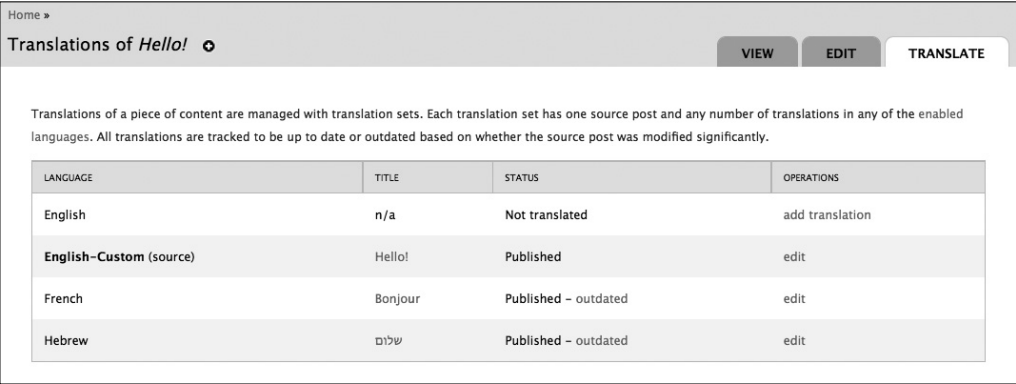


Рис. 19.32. Исходное сообщение отредактировано, а перевод сообщения помечен как устаревший

Исходная нода и ее перевод имеют разные номера нод и фактически существуют как совершенно разные ноды в базе данных. Они связаны друг с другом столбцом `tnid` таблицы `node`, который содержит идентификатор исходной ноды. Если английская версия является исходной, и к тому же первой нодой на сайте, а переводы на французский и иврит — следующие добавленные ноды, таблица нод будет выглядеть, как показано на рис. 19.33.

		nid The primary identifier for a node.	vid The current node revision version identifier.	type The type of this node.	language The language of this node.	title The title of this node, always treated as non-markup plain text.	uid The user id that owns this node; initially, this is the user that created it.	status Boolean indicating whether the node is published (visible to non-administrators).	created The Unix timestamp when the node was created.	changed The Unix timestamp when the node was most recently saved.	comment Whether comments are allowed on this node: 0 = no, 1 = closed (read only), 2 = open (read/write).	promote Boolean indicating whether the node should be displayed at the top of lists in which it appears.	sticky Boolean indicating whether the node should be displayed at the top of lists in which it appears.	tnid The translation set id for this node, which equals the node id of the source post in each set.	translate A boolean indicating whether this translation page needs to be updated.
☐		15	15	page	en-US	Hello!	1	1	1281286449	1281302188	1	0	0	15	0
☐		16	16	page	fr	Bonjour	1	1	1281287176	1281287176	1	0	0	15	1
☐		17	17	page	he	שלום	1	1	1281302016	1281302016	1	0	0	15	1

Рис. 19.33. Столбец `tnid` отслеживает взаимосвязи между исходными нодами и их переводами

Обратите внимание, что устаревший перевод обозначается значением 1 в столбце `translate`.

Файлы, относящиеся к локализации и переводу

Иногда разобраться в том, какие части Drupal отвечают за те или иные функции локализации и перевода, довольно непросто. Эти файлы и выполняемые ими функции перечислены в табл. 19.1.

Таблица 19.1. Файлы, относящиеся к локализации и переводу в среде Drupal

Файл	Функции
<code>includes/bootstrap.inc</code>	Запускает фазу <code>DRUPAL_BOOTSTRAP_LANGUAGE</code> , определяющую текущий язык
<code>includes/language.inc</code>	Включается процессом начальной загрузки при наличии нескольких включенных языков. Предоставляет код для выбора языка и перезаписи внутренних URL в соответствии с выбранным языком
<code>includes/common.inc</code>	Содержит функцию <code>t()</code> , а также <code>drupal_add_css()</code> , которая поддерживает языки с письмом справа налево

Файл	Функции
<code>includes/locale.inc</code>	Содержит интерфейс пользователя и функции для управления языковыми переводами
<code>modules/locale/locale.module</code>	Предоставляет замену строк и импорт переводов, когда модули или темы установлены и активированы. Добавляет интерфейс языковых настроек в путь, ноды и формы типов нод
<code>modules/translation/translation.module</code>	Управляет исходными нодами и их переводами
<code>modules/translation/translation.admin.inc</code>	Предоставляет обзор переводов, отображаемый при щелчке на вкладке Translate (см. рис. 19.31)

Дополнительные ресурсы

Поддержка интернационализации — важнейшая часть проекта Drupal. Следить за продвижением в этом направлении или принять участие в этой работе можно на странице <http://groups.drupal.org/i18n>. Посетите также страницу модуля Translation Management по адресу http://drupal.org/project/translation_management.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Знать, как работает функция `t()`.
- Уметь настраивать встроенные строки Drupal.
- Уметь экспортировать собственные настройки.
- Знать, что собой представляют файлы переносимых объектов и шаблонов переносимых объектов.
- Загружать файлы шаблонов переносимых объектов и создавать собственные.
- Уметь импортировать существующий перевод Drupal.
- Знать, как использовать таблицы стилей для поддержки языков с письмом справа налево.
- Знать, каким образом настройки согласования языков влияют на Drupal.
- Знать, как работает перевод контента.

ГЛАВА 20

Протокол XML-RPC

Среда Drupal хорошо взаимодействует с другими системами. Это означает, что если существует какой-то открытый стандарт, есть вероятность, что Drupal поддерживает его либо изначально, либо посредством дополнительного модуля. Стандарт XML-RPC не является исключением из этого правила — Drupal поддерживает его изначально. В этой главе вы узнаете, как эффективно воспользоваться возможностями Drupal по отправке и получению вызовов XML-RPC.

Что собой представляет XML-RPC

Вызов удаленной процедуры (remote procedure call) — это обращение одной программы к другой с запросом на выполнение какой-то функции. XML-RPC — стандарт вызовов удаленных процедур, в соответствии с которым запрос кодируется с использованием XML и передается с помощью протокола HTTP. Протокол XML-RPC был разработан Дейвом Винером (Dave Winer) из компании UserLand Software в сотрудничестве с Microsoft (см. www.xmlrpc.com/spec). Этот протокол предназначен специально для распределенных веб-систем, взаимодействующих между собой, подобных ситуациям, когда один Drupal-сайт запрашивает у другого какую-то информацию.

Взаимодействие по протоколу XML-RPC осуществляется между двумя участниками. Один из них — сайт, с которого выполняется запрос — называется *клиентом*. Сайт, который принимает запрос, является *сервером*.

Предварительные требования для протокола XML-RPC

Если сайт будет действовать только в качестве сервера, беспокоиться не о чем, поскольку входящие запросы XML-RPC используют стандартный веб-порт (обычно с номером 80). Файл `xmlrpc.php` в инсталляции Drupal содержит код, который запускается для обслуживания входящего запроса XML-RPC. Этот элемент называют конечной точкой (endpoint) XML-RPC.

На заметку! Некоторые пытаются повысить уровень безопасности, переименовывая файл `xmlrpc.php`, чтобы изменить свою конечную точку RPC-XML. Это препятствует зондированию интерфейсов XML-RPC сервера вредоносными роботами. Другие вообще удаляют этот файл, если сайт не принимает запросы XML-RPC.

Чтобы сайт Drupal работал в качестве клиента, он должен иметь возможность отправлять исходящие запросы HTTP. Некоторые компании хостинга запрещают это по соображениям безопасности, и попытки отправки таких запросов не проходят через их брандмауэры.

Клиенты XML-RPC

Клиент — это компьютер, который будет посылать запрос. Он отправляет серверу стандартный HTTP-запрос POST. Тело этого запроса сформировано из кода XML и содержит единственный дескриптор `<methodCall>`. В него вложены два дескриптора — `<methodName>` и `<params>`. Рассмотрим работу клиента на практическом примере.

На заметку! В контексте XML вызываемую удаленную процедуру называют методом. Именно потому при кодировании XML имя такой процедуры помещается в дескриптор `<methodName>`.

Пример клиента XML-RPC: получение текущего времени

Сайт, содержащий спецификацию XML-RPC (www.xmlrpc.com), предлагает также несколько тестовых реализаций. В качестве первого примера запросим у этого сайта текущее значение времени посредством протокола XML-RPC:

```
$time = xmlrpc('http://time.xmlrpc.com/RPC2',
    array('currentTime' => array()));
```

В этом примере мы вызываем Drupal-функцию `xmlrpc()`, предлагая ей установить контакт с путем `RPC2`, определенным на сервере `time.xmlrpc.com`, и потребовать от этого сервера выполнения метода `currentTime.getCurrentTime()`. Никакие параметры вызову не передаются. Drupal преобразует все это в HTTP-запрос, который выглядит подобно показанному ниже:

```
POST /RPC2 HTTP/1.0
Host: time.xmlrpc.com
User-Agent: Drupal (+http://drupal.org/)
Content-Length: 118
Content-Type: text/xml

<?xml version="1.0"?>
<methodCall>
  <methodName>currentTime.getCurrentTime</methodName>
  <params></params>
</methodCall>
```

Сервер `time.xmlrpc.com` успешно запускает эту функцию и возвращает следующий ответ:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 183
Content-Type: text/xml
Date: Wed, 23 Apr 2008 16:14:30 GMT
Server: UserLand Frontier/9.0.1-WinNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <dateTime.iso8601>20080423T09:14:30</dateTime.iso8601>
      </value>
    </param>
  </params>
</methodResponse>
```

Когда ответ возвращается, Drupal анализирует его и распознает в нем единственное значение международного формата даты ISO 8601. Затем Drupal возвращает не только представление времени и даты в формате ISO 8601, но также все составляющие времени: год, месяц, день, часы, минуты и секунды. Объект с этими свойствами присваивается переменной `$time`, как показано на рис. 20.1.

Здесь важно уяснить следующие моменты.

- Мы обратились к удаленному серверу, и он ответил.
- Запрос и ответ были представлены в формате XML.
- Мы воспользовались функцией `xmlrpc()` и указали URL, а также имя удаленной процедуры, которую нужно вызвать.
- Возвращаемое значение было помечено как определенный тип данных.
- Среда Drupal распознала тип данных и автоматически проанализировала ответ.
- Все это было выполнено с помощью единственной строки кода.

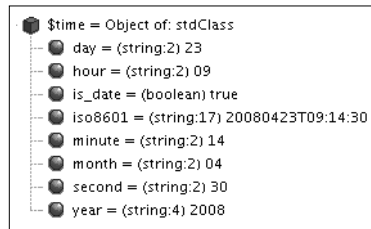


Рис. 20.1. Результат вызова XML-RPC для получения текущего значения времени

Пример клиента XML-RPC: получение названия штата

Рассмотрим несколько более сложный пример. Он сложнее только потому, что наряду с именем вызываемого удаленного метода передается также параметр. На своем сайте `betty.userland.com` компания UserLand Software поддерживает веб-службу, которая выводит список 50 штатов США в алфавитном порядке. Так, если вы запросить штат 1, она возвращает Alabama (Алабама), а штат 50 — Wyoming (Вайоминг). Используемый при этом метод имеет имя `examples.getStateName`. Запросим информацию о штате с номером 3:

```
$state_name = xmlrpc('http://betty.userland.com/RPC2',
    array('examples.getStateName' => array(3)));
```

В результате переменной `$state_name` присваивается значение Arizona. XML-код, отправляемый системой Drupal (для простоты HTTP-заголовки опущены), имеет следующий вид:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value>
        <int>3</int>
      </value>
    </param>
  </params>
</methodCall>
```

Сайт `betty.userland.com` возвращает следующий ответ:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>Arizona</value>
    </param>
  </params>
</methodResponse>
```

Обратите внимание, что Drupal автоматически определяет отправляемый параметр как целочисленный и именно так кодирует его в запросе. Но что происходит в ответе? Это значение не окружено никакими дескрипторами типа! Разве оно не должно выглядеть как `<value><string>Arizona</string></value>?` Да, это также работало бы, однако в XML-RPC значения, для которых тип не указан, считаются строковыми, поэтому можно использовать приведенный более лаконичный код. Осталось выполнить клиентский запрос XML-RPC в Drupal — для этого достаточно одной строки:

```
$result = xmlrpc($url,
    array($method => array($param_1, $param_2, $param_3...)), $options);
```

Обработка ошибок клиента XML-RPC

При взаимодействии с удаленными серверами многое может пойти не так, как планировалось. Например, в коде может присутствовать синтаксическая ошибка, сервер может находиться в автономном режиме или же может не работать сеть. Посмотрим, как Drupal справляется с ситуациями подобного рода.

Ошибки сети

Drupal использует функцию `drupal_http_request()` из `includes/common.inc` для выдачи исходящих HTTP-запросов, в том числе и запросов XML-RPC. Внутри этой функции применяется PHP-функция `fsockopen` для открытия сокета к удаленному серверу. Если сокет не может быть открыт, Drupal либо установит отрицательный код ошибки, либо код 0, в зависимости от того, на какой платформе работает PHP, и в какой момент открытия сокета произошла ошибка. Давайте намеренно неправильно укажем имя сервера при получении имени штата:

```
$url = 'http://betty.userland.comm/RPC2';
$method = 'examples.getStateName';
$state_name = xmlrpc($url, array($method => array(3)));
if ($error = xmlrpc_error()) {
    if ($error->code <= 0) {
        $error->message = t('Outgoing HTTP request failed because the socket
could not be opened.');
```

Это приведет к отображению следующего сообщения:

```
Could not get state name because the remote site gave an error: Outgoing
HTTP request failed because the socket could not be opened. (-19891355).
```

Невозможно получить имя штата, потому что удаленный сервер сообщил об ошибке: Исходящий HTTP-запрос был безуспешным, поскольку не удалось открыть сокет. (-19891355).

Ошибки HTTP

Приведенный код подойдет для обработки ошибок HTTP, таких как ситуация, когда сервер работает, но никакой веб-службы по указанному пути не обнаружено. Следующий код предлагает drupal.org запустить веб-службу, а drupal.org сообщает, что по адресу `http://drupal.org/RPC2` ничего нет:

```
$state = xmlrpc('http://drupal.org/RPC2', array('examples.getStateName'));
if ($error = xmlrpc_error()) {
```



```

if ($error->code <= 0) {
    $error->message = t('Outgoing HTTP request failed because the socket
could not be opened.');
```

```

    }
    drupal_set_message(t('Could not get state name because the remote site
gave an error: %message (@code).', array(
    '%message' => $error->message,
    '@code' => $error->code
    )
    )
);

```

Это приведет к отображению следующего сообщения:

```

Could not get state name because the remote site gave an error: Not Found (404)
Невозможно получить имя штата, потому что удаленный сайт выдал ошибку: Не
найдено (404)

```

Ошибки синтаксиса вызова

Ниже показано, что будет возвращено в случае успешного достижения сервера `betty.userland.com`, но при попытке получения от него имени штата без указания номера, который является обязательным параметром:

```

$state_name = xmlrpc('http://betty.userland.com/RPC2',
    array('examples.getStateName'));
```

Удаленный сервер вернет следующее:

```

<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value>
            <int>7</int>
          </value>
        </member>
        <member>
          <name>faultString</name>
          <value>
            <string>Can't evaluate because the name "0" hasn't been defined.
            </string>
          </value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>

```

Сервер работает, и обмен данными с ним выполняется успешно; приведенный код возвращается вместе с HTTP-кодом ответа 200 ОК. Ошибка определяется по коду ошибки и строке описания ошибки в XML-ответе. Код обработки ошибки должен быть тем же:

```

$state_name = xmlrpc('http://betty.userland.com/RPC2',
    array('examples.getStateName'));
if ($error = xmlrpc_error()) {
    if ($error->code <= 0) {

```

```

$error->message = t('Outgoing HTTP request failed because the socket
could not be opened.');
```

```

}
drupal_set_message(t('Could not get state name because the remote site
gave an error: %message (@code).', array(
  '%message' => $error->message,
  '@code' => $error->code
))
);

```

Этот код приведет к отображению следующего сообщения:

Could not get state name because the remote site gave an error: *Can't evaluate the expression because the name "0" hasn't been defined. (7).*

Не удалось получить имя штата, поскольку удаленный сайт выдал ошибку: Невозможно проанализировать выражение, т.к. имя "0" не определено. (7).

Обратите внимание, что при сообщении об ошибках необходимо отразить следующие три момента: попытка выполнения какого действия была предпринята, почему она оказалась неудачной, и предоставить дополнительную информацию, к которой имеется доступ. Часто лаконичное, понятное пользователю сообщение отображается с помощью функции `drupal_set_message()`, чтобы просто уведомить пользователя о случившемся, а более подробное сообщение записывается в систему наблюдения (watchdog) и может быть просмотрено на странице Reports⇒Recent log messages (Отчеты⇒Последние сообщения в журнале).

Простой сервер XML-RPC

Как было показано в примерах клиента XML-RPC, система Drupal выполняет большую часть самой трудной работы. Теперь рассмотрим простой пример сервера. Чтобы установить сервер, необходимо выполнить три действия.

1. Определить функцию, которую нужно выполнять при поступлении запроса от клиента.
2. Отобразить эту функцию именем общедоступного метода.
3. Определить сигнатуру метода (необязательно).

Как обычно для Drupal, нужно хранить код отдельно от ядра системы и просто включать его в систему в качестве модуля. Ниже приведен код небольшого модуля, выводящего приветственное сообщение посредством XML-RPC. Создайте файл `sites/all/modules/custom/remotehello/remotehello.info` со следующим содержимым:

```

name = Remote Hello
description = Greets XML-RPC clients by name.
package = Pro Drupal Development
core = 7.x

```

А вот код `remotehello.module`:

```

<?php

/**
 * Реализация hook_xmlrpc().
 * Отображение внешних имен методов XML-RPC функциями обратного вызова PHP.
 */

```

```

function remotehello_xmlrpc() {
    $methods['remoteHello.hello'] = 'xmls_remotehello_hello';
    return $methods;
}

/**
 * Приветствие пользователя.
 */
function xmls_remotehello_hello($name) {
    if (!$name) {
        return xmlrpc_error(1, t('I cannot greet you by name if you do not
        provide one.'));
    }
    return t('Hello, @name!', array('@name' => $name));
}

```

Отображение метода с помощью `hook_xmlrpc()`

Хук `xmlrpc` описывает внешние методы XML-RPC, предоставляемые данным модулем. В рассматриваемом примере предоставляется только один метод. В данном случае именем метода является `remoteHello.hello`. Это имя будет использовать запрашивающая сторона, и оно может быть совершенно произвольным. Рекомендуется строить имя в виде строки с разделителем-точкой, используя имя модуля в качестве первой части и описатель — в качестве последней.

На заметку! Хотя в общем случае в Drupal имена в стиле “camelCase” (“верблюжий” стиль) не приняты, внешние имена методов XML-RPC являются исключением.

Вторая часть массива — имя функции, которая будет вызываться при поступлении запроса на `remoteHello.hello`. В нашем примере вызывается PHP-функция `xmls_remotehello_hello()`. Поскольку речь идет о разработке модулей, придется создавать много функций. Добавление к имени функции приставки `xmls` (сокращение от XML-RPC Server) позволит с первого взгляда установить, что данная функция взаимодействует с внешним миром. Аналогично `xmls` можно использовать для функций, которые обращаются к другим сайтам. В частности, следование этому соглашению удобно при создании модуля, который, по сути, обращается к самому себе.

Если модуль должен определять наличие ошибки, используйте функцию `xmlrpc_error()` для определения кода ошибки и построения строки, описывающей то, что произошло на стороне клиента. Числовые коды ошибок произвольны и специфичны для конкретных приложений.

Предполагая, что сайт с данным модулем находится в `example.com`, свое имя можно отправить из отдельной инсталляции Drupal (например, `example2.com`), используя следующий код:

```

$url = 'http://example.com/xmlrpc.php';
$method_name = 'remoteHello.hello';
$name = t('Joe');
$result = xmlrpc($url, array($method_name => array($name)));

```

Результат выполнения кода:

```
$result is now "Hello, Joe."
```

Автоматическая проверка типа параметра с помощью `hook_xmlrpc()`

Хук `xmlrpc` имеет две формы. В простой форме, которая показана в примере `remotehello.module`, он просто отображает внешнее имя метода PHP-функцией. В более сложной форме он описывает сигнатуру метода, т.е. сообщает типы XML-RPC возвращаемого значения и каждого из параметров (список типов доступен по адресу www.xmlrpc.com/spec). Более сложная форма хука `xmlrpc` для `remotehello.module` имеет следующий вид:

```
/**
 * Реализация hook_xmlrpc().
 * Отображает внешние имена методов XML-RPC функциями обратного вызова.
 * Многословный синтаксис, со спецификацией типов данных возвращаемого
 * значения и параметров.
 */
function remotehello_xmlrpc() {
  $methods = array();
  $methods[] = array(
    'remoteHello.hello', // Внешнее имя метода.
    'xmls_remotehello_hello', // PHP-функция, которая должна быть выполнена.
    array('string', 'string'), // Тип возвращаемого значения, за которым
                              // следуют типы любых используемых параметров.
    t('Greets XML-RPC clients by name.') // Описание.
  );
  return $methods;
}
```

На рис. 20.2 показан жизненный цикл запроса XML-RPC на пути от клиента XML-RPC до нашего модуля. Реализация хука `xmlrpc` для модуля с применением его более сложной формы предоставит несколько преимуществ. Во-первых, Drupal автоматически будет проверять поступающие типы на соответствие сигнатуре метода и, если проверка завершится неудачей, возвратит клиенту сообщение `-32602: Server error. Invalid method parameters` (Ошибка сервера. Недопустимые параметры метода). (Это означает также повышение избирательности функции — т.е. отсутствие какого-либо приведения типов, как в случае принятия строкового значения '3' вместо целочисленного значения 3!) Во-вторых, при использовании более сложной формы хука `xmlrpc` встроенные XML-RPC-методы Drupal `system.methodSignature` и `system.methodHelp` будут возвращать информацию о методе. Обратите внимание, что описание, предоставленное в реализации хука `xmlrpc`, будет возвращено в качестве текста справки в методе `system.methodHelp`, поэтому позаботьтесь о понятном описании.

Встроенные методы XML-RPC

Среда Drupal поставляется с несколькими готовыми методами XML-RPC. Эти встроенные методы описаны в последующих разделах.

`system.listMethods`

Метод `system.listMethods` перечисляет доступные методы XML-RPC. Ниже приведен ответ, который возвращает сайт Drupal при запросе предоставляемых им методов.

```
// Получение массива всех методов XML-RPC, доступных на данном сервере.
$url = 'http://example.com/xmlrpc.php';
$methods = xmlrpc($url, array('system.listMethods'));
```

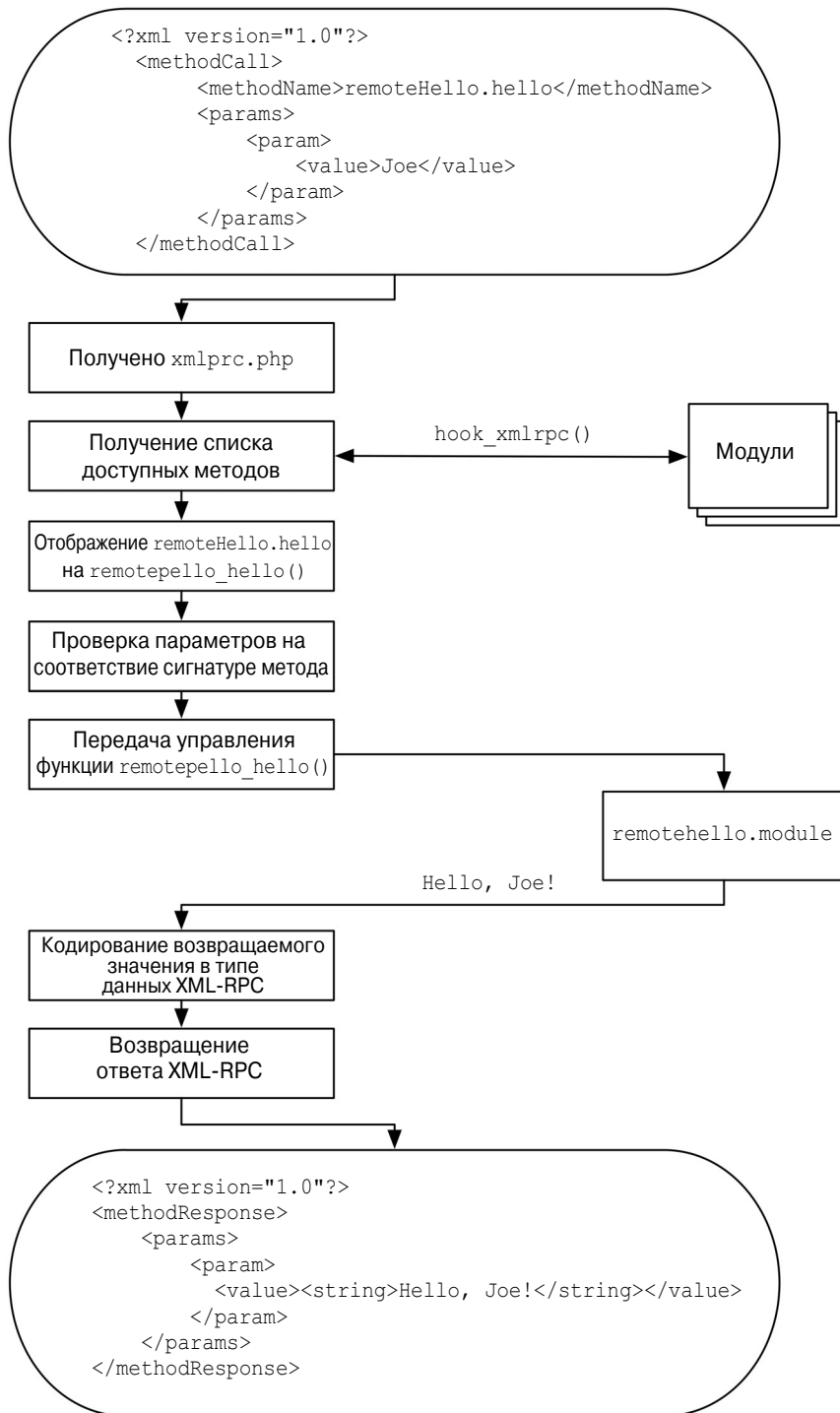


Рис. 20.2. Обработка входящего запроса XML-RPC

Ответ сервера выглядит следующим образом:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <array>
          <data>
            <value>
              <string>system.multicall</string>
            </value>
            <value>
              <string>system.methodSignature</string>
            </value>
            <value>
              <string>system.getCapabilities</string>
            </value>
            <value>
              <string>system.listMethods</string>
            </value>
            <value>
              <string>system.methodHelp</string>
            </value>
            <value>
              <string>remoteHello.hello</string>
            </value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodResponse>
```

Теперь содержимое `$methods` представляет собой массив имен методов, доступных на сервере:

```
('system.multicall', 'system.methodSignature', 'system.getCapabilities',
'system.listMethods', 'system.methodHelp', 'remoteHello.hello')
```

system.methodSignature

Этот встроенный метод XML-RPC возвращает массив типов данных. Вначале выводится тип данных возвращаемого функцией значения, а затем — любых параметров, ожидаемых данным методом. Например, метод `remoteHello.hello` возвращает строку и принимает один параметр — строку, содержащую имя клиента. Вызовем `system.methodSignature`, чтобы выяснить, устроит ли это Drupal:

```
// Получение сигнатуры метода для нашего примера.
$url = 'http://example.com/xmlrpc.php';
$signature = xmlrpc($url, 'system.methodSignature', array('remoteHello.hello'));
```

Как и следовало ожидать, значением `$signature` становится массив: `('string', 'string')`.

system.methodHelp

Этот встроенный метод XML-RPC Drupal возвращает описание метода, определенно-го в реализации хука `xmlrpc` модуля, предоставляющего этот метод:

```
// Получение строки справки для нашего примера метода.
$url = 'http://example.com/xmlrpc.php';
$help = xmlrpc($url, 'system.methodHelp', array('remoteHello.hello'));
```

Теперь значением `$help` является строка, приветствующая клиентов XML-RPC по имени.

system.getCapabilities

Этот встроенный метод XML-RPC описывает возможности сервера XML-RPC в терминах реализуемых им спецификаций. Drupal реализует следующие спецификации:

```
xmlrpc:
specURL      http://www.xmlrpc.com/spec
specVersion  1

faults_interop:
specURL      http://xmlrpc-epi.sourceforge.net/specs/rfc.fault_codes.php
specVersion  20010516

system.multicall
specURL      http://web.archive.org/web/20101015050132/http://www.xmlrpc.com/discuss/msgReader$1208
specVersion  1

introspection
specURL      http://scripts.incutio.com/xmlrpc/introspection.html
specVersion  1
```

system.multiCall

Еще одним заслуживающим внимания встроенным методом является `system.multiCall`, который позволяет осуществлять вызов более одного метода XML-RPC в одном HTTP-запросе. За дополнительной информацией об этом соглашении (которое не нашло отражения в спецификации XML-RPC) обратитесь по следующему URL-адресу: [http://web.archive.org/web/20060502175739/http://www.xmlrpc.com/discuss/msgReader\\$1208](http://web.archive.org/web/20060502175739/http://www.xmlrpc.com/discuss/msgReader$1208).

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Отправлять XML-RPC-вызовы с сайта Drupal на другой сервер.
- Понимать, как Drupal отображает методы XML-RPC на PHP-функции.
- Уметь реализовывать простую и сложную версии хука `xmlproc`.
- Знать встроенные методы XML-RPC в Drupal.

ГЛАВА 21

Написание защищенного кода

Едва ли не ежедневно приходится читать сообщения о том, что в том или ином программном обеспечении обнаружены проблемы с безопасностью. Защита веб-приложения и сервера от нежелательных гостей должна быть одним из главных приоритетов для любого серьезного разработчика.

Существует множество способов, с помощью которых злонамеренный пользователь может попытаться дискредитировать Drupal-сайт. Некоторые из них основаны на внедрении в систему исполняемого кода, который позволит управлять данными в базе данных, просматривать материалы, к которым пользователь не должен иметь доступа, и отправлять нежелательную электронную почту через установленную копию Drupal. В этой главе будет показано, как выполнять программирование, чтобы противостоять атакам подобного рода.

К счастью, Drupal предлагает ряд инструментальных средств, которые облегчают устранение самых распространенных причин появления брешей в системе безопасности.

Обработка вводимой пользователем информации

Как правило, пользователи взаимодействуют с Drupal посредством наборов форм, таких как форма отправки данных ноды или форма отправки комментария. Пользователи могут также удаленно отправить сообщение в блог Drupal посредством XML-RPC с помощью модуля `blogapi` (<http://drupal.org/project/blogapi>). Подход Drupal к вводимой пользователем информации можно сформулировать следующим образом: *сохранять исходные данные, но фильтровать выводимые*. База данных всегда должна содержать в точности то, что ввел пользователь. В процессе подготовки введенной пользователем информации к ее отображению на веб-странице она очищается (т.е. потенциально исполняемый код нейтрализуется).

Бреши в системе безопасности могут появиться, когда текст, введенный пользователем, не очищен и будет исполняемым внутри кода. Это происходит, если при создании программы все возможности не были продуманы в полной мере. Например, иногда ожидают, что пользователи будут вводить только стандартные символы, в то время как фактически они могут вводить нестандартные строки или закодированные символы, такие как управляющие символы. Возможно, вам приходилось встречать URL с символами `%20`, например, `http://example.com/my%20document.html`. Это символ пробела, который был закодирован в соответствии со спецификацией URL (см. www.w3.org/Addressing/URL/url-spec.html). Когда кто-то сохраняет файл с именем `my document.html` и эта операция обслуживается веб-сервером, такой пробел кодируется.

Символ % обозначает кодированный символ, а 20 указывает, что это — ASCII-символ 32 (20 — шестнадцатеричное представление десятичного числа 32). Хитроумное использование закодированных символов злоумышленниками может создать проблемы, как будет показано далее в этой главе.

Соображения по поводу типов данных

Имея дело с текстом в такой системе, как Drupal, где вводимая пользователем информация отображается в виде части веб-сайта, целесообразно рассматривать вводимые пользователем данные как типизированную переменную. Если вам доводилось программировать на языке со строгим контролем типов, таком как Java, вы должны быть знакомы с типизированными переменными. Например, в Java целое число — действительно целое число, и оно не будет обрабатываться как строка, если только программист явно не выполнит преобразование. В PHP (языке со слабым контролем типов) целое число вполне может трактоваться как строка или как целое число, в зависимости от контекста, благодаря возможности автоматического преобразования типов, доступной в PHP. Тем не менее, опытные программисты PHP тщательно заботятся о типах и используют автоматическое преобразование лишь для достижения конкретных целей. Аналогично, даже если вводимые пользователем данные из, скажем, поля Body (Тело) формы отправки данных ноды могут трактоваться как текст, намного лучше считать их *определенным типом текста*. Вводит ли пользователь простой текст? Или, возможно, он вводит HTML-дескрипторы и ожидает, что они будут визуализированы? Если это так, то не могут ли среди них присутствовать опасные дескрипторы, такие как код JavaScript, который заменяет страницу рекламой рингтонов мобильных телефонов? Страница, которая будет отображена на компьютере пользователя, имеет формат HTML. Вводимые пользователем данные — это разнообразные “типы” текстовых форматов, которые должны быть безопасно преобразованы в HTML прежде, чем текст будет отображен. Такой подход к вводимой пользователем информации помогает понять, как работают функции преобразования текста Drupal. Обычно используемые типы текстового ввода, наряду с функциями, преобразующими текст в другой формат, перечислены в табл. 21.1.

Таблица 21.1. Безопасные преобразования одного типа текста в другой

Исходный формат	Целевой формат	Функция Drupal	Действие
Простой текст	HTML	<code>check_plain()</code>	Кодирует специальные символы в HTML-объекты и проверяет строки UTF-8 для предотвращения межсайтовых атак посредством сценариев в браузерах Internet Explorer 6
HTML-текст	HTML	<code>filter_xss()</code>	Удаляет символы и конструкции, которые могут “обманывать” браузеры. Гарантирует, что все HTML-объекты правильно сформированы. Гарантирует, что все дескрипторы и атрибуты HTML правильно сформированы, и никакие дескрипторы HTML не содержат URL-адреса с запрещенным протоколом (например, JavaScript)

Исходный формат	Целевой формат	Функция Drupal	Действие
Форматированный текст	HTML	<code>check_markup()</code>	Пропускает текст через все включенные фильтры
Простой текст	URL	<code>drupal_encode_path()</code>	Кодирует путь Drupal для использования в URL
URL HTML		<code>check_url()</code>	Отсекает опасные протоколы, такие как <code>javascript:</code>
Простой текст	MIME	<code>mime_header_encode()</code>	Преобразует символы, не соответствующие стандарту ASCII, в символы, закодированные по стандарту UTF-8

Простой текст

Простой текст — это текст, который не содержит информации о форматировании. Например, если вы просите пользователя указать в форме предпочитаемый цвет, то ожидаете в качестве ответа что-то вроде `green` (зеленый) или `purple` (фиолетовый), без какой-либо разметки. Вставка этой введенной пользователем информации в другую веб-страницу без проверки того, что она действительно содержит только простой текст — большой риск в смысле безопасности. Например, вместо простого указания цвета пользователь может ввести следующее:

```

http://evil.example.com/133/index.php?s=11&</a>;ce_cid=38181161'">
```

Доступная функция `check_plain()` позволяет нейтрализовать все другие символы за счет их кодирования как объектов HTML. Текст, возвращенный функцией `check_plain()`, не будет содержать никаких HTML-дескрипторов, поскольку все они будут преобразованы в объекты. Если пользователь введет вредоносный код JavaScript, функция `check_plain()` превратит его в следующий текст, который будет совершенно безобидным при визуализации в HTML:

```
&lt;img src=&quot;javascript:window.location =&#039;&lt;a
href=&quot;http://evil.example.com/133/index.php?s=11&amp;&quot;&gt;http://evil.
example.com/133/index.php?s=11&amp;&lt;/a>;ce_cid=38181161&#039;&quot;&gt;
```

HTML-текст

HTML-текст может содержать разметку HTML. Однако никогда не следует слепо полагаться на то, что пользователи будут вводить только “безопасный” HTML-код. В общем случае пользователям нужно разрешать применять лишь определенное подмножество дескрипторов HTML. Например, дескриптор `<script>` не относится к числу тех, применение которых можно разрешать во всех случаях, поскольку он позволяет пользователям по своему выбору запускать сценарии на сайте. Аналогично, не следует позволять пользователям использовать дескриптор `<form>` для установки форм на сайте.

Форматированный текст

Форматированный текст — это текст, который содержит больше информации, чем простой текст, но не обязательно представленный в формате HTML. Он может содержать вики-разметку, код досок объявлений Bulletin Board Code (BBCode) или код на каком-то

другом языке разметки. Перед отображением такой текст должен быть пропущен через фильтр для преобразования разметки в формат HTML.

На заметку! Подробнее фильтры описаны в главе 11.

URL

URL — это URL-адрес, который был построен на основе введенных пользователем данных или другого непроверенного источника. Например, мы могли ожидать, что пользователь введет `http://example.com`, но вместо этого он ввел `javascript:runevilJS()`. Прежде чем отображать URL на HTML-странице, его нужно проверить функцией `check_url()`, чтобы удостовериться в его корректности и отсутствии угрозы атаки.

Использование функций `check_plain()` и `t()` для очистки вывода

Применяйте функцию `check_plain()` всякий раз, когда имеете дело с текстом, которому не доверяете и не хотите, чтобы в нем присутствовала какая-либо разметка.

Ниже показан элементарный способ обработки введенной пользователем информации, исходящий из того, что пользователь ввел свой любимый цвет в текстовом поле. Следующий код небезопасен:

```
drupal_set_message("Your favorite color is $color!"); // Отсутствует
// проверка ввода!
```

А вот пример безопасного, но далекого от идеала кода:

```
drupal_set_message('Your favorite color is ' . check_plain($color));
```

Это код неудачен, потому что мы получаем текстовую строку (а именно — неявный результат вызова функции `check_plain()`), не включенную в функцию `t()`, которая всегда должна использоваться для текстовых строк. При написании кода, подобного приведенному выше, будьте готовы к нареканиям со стороны переводчиков, которые не смогут перевести фразу, поскольку она не обрабатывается функцией `t()`.

Нельзя просто заключить переменные в двойные кавычки и передать их функции `t()`. Следующий код все еще небезопасен, поскольку в нем не используются заполнители:

```
drupal_set_message(t("Your favorite color is $color!")); // Отсутствует
// проверка ввода!
```

Функция `t()` предоставляет встроенный способ превращения строк в безопасные за счет применения маркера заполнителя с односимвольным префиксом. Следующий код соответствует требованиям безопасности и имеет приемлемую форму:

```
drupal_set_message(t('Your favorite color is @color', array('@color' => $color)));
```

Обратите внимание, что ключ в массиве (`@color`) совпадает с маркером заполнителя в строке. Это приводит к передаче следующего сообщения:

```
Your favorite color is brown.
```

Префикс `@` указывает функции `t()` на необходимость замены маркера значением, полученным из `check_plain()`.

На заметку! При переводе Drupal маркер обрабатывается функцией `check_plain()`, а переведенная строка — нет. Таким образом, вы должны доверять своим переводчикам.

В данном примере можно выделить сделанный пользователем выбор цвета, изменив стиль представления значения цвета. Для этого служит префикс %, который означает “выполнить `-theme('placeholder', $value)` применительно к значению”. Значение будет неявно обработано функцией `check_plain()`, как показано на рис. 21.1. Префикс % применяется очень часто.

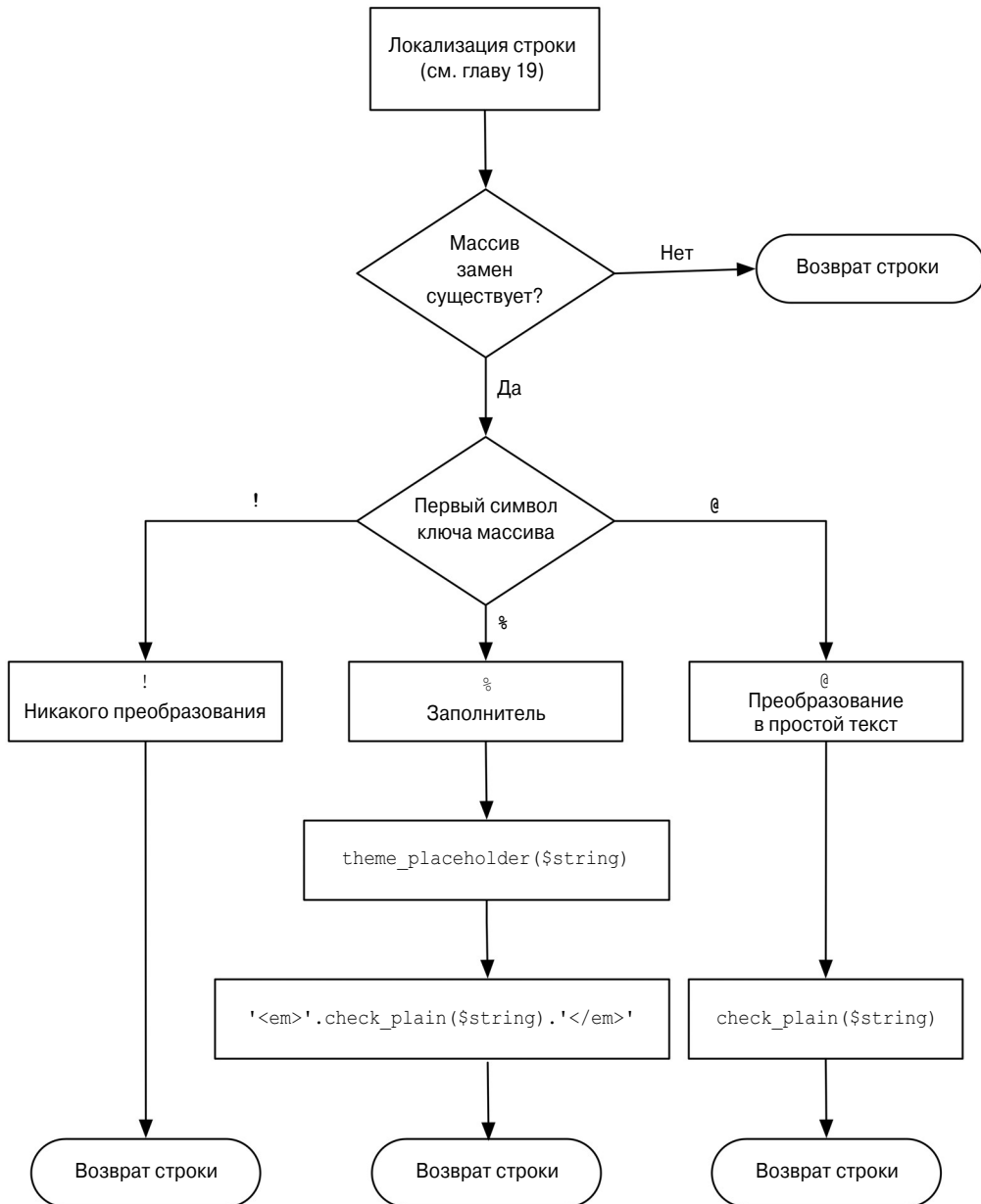


Рис. 21.1. Влияние префиксов заполнителя на замену строки

Следующий код соответствует требованиям безопасности и достаточно оптимален:

```
drupal_set_message(t('Your favorite color is %color', array('%color' => $color)));
```

В результате отобразится сообщение, подобное показанному ниже. Кроме отмены значения функция `theme_placeholder()` помещает его между дескрипторами `<em></em>`.

```
Your favorite color is brown.
```

При наличии текста, который был предварительно очищен, проверки внутри `t()` можно отключить с помощью префикса `!`. Например, функция `l()` строит ссылку и для удобства в процессе ее построения подвергает текст ссылки обработке с помощью функции `check_plain()`. Поэтому в следующем примере можно безопасно использовать префикс `!`:

```
// Функция l() пропускает текст через check_plain() и возвращает очищенный текст,
// поэтому можно не выполнять check_plain($link) или делать это посредством t().
$link = l($user_supplied_text, $path);
drupal_set_message(t('Go to the website !website', array('!website' => $link)));
```

На заметку! Функция `l()` обеспечивает обработку текста ссылки функцией `check_plain()`, если только вы не указали функции `l()`, что текст уже имеет формат HTML, установив `html` в значение `TRUE` в параметре `options`. См. <http://api.drupal.org/api/function/l/7>.

Влияние заполнителей `@`, `%` и `!` на замену строки в функции `t()` продемонстрировано на рис. 21.1. Хотя для простоты это и не показано на рисунке, помните, что можно использовать несколько заполнителей, определяя их в строке и добавляя члены в массив, например:

```
drupal_set_message(t('Your favorite color is %color and you like %food',
  array('%color' => $color, '%food' => $food)));
```

Применяя префикс `!`, нужно быть особенно внимательными, поскольку это означает, что строка не будет обработана функцией `check_plain()`.

Использование функции `filter_xss()` для предотвращения межсайтовых атак с помощью сценариев

Межсайтовые сценарии (Cross Site Scripting — XSS) — часто применяемый метод атаки на веб-сайт, при котором атакующий получает возможность вставлять в веб-страницу собственный код, который затем может использоваться для причинения ущерба.

На заметку! Примеры XSS-атак рассмотрены на сайте <http://ha.ckers.org/xss.html>.

Предположим, что на своем веб-сайте вы разрешаете пользователям вводить HTML-код, ожидая, что они будут вводить что-то вроде:

```
<em>Hi!</em> My name is Sally, and I...
```

но вместо этого они вводят:

```
<script src=http://evil.example.com/xss.js"></script>
```

Еще раз напоминаем: никогда не доверяйте информации, введенной пользователем! Вот сигнатура функции `filter_xss()`:

```
filter_xss($string, $allowed_tags = array('a', 'em', 'strong', 'cite',
  'blockquote', 'code', 'ul', 'ol', 'li', 'dl', 'dt', 'dd'))
```

Функция `filter_xss()` выполняет следующие операции с передаваемой ей текстовой строкой.

1. Проверяет, является ли фильтруемый текст допустимым с точки зрения кодировки UTF-8, чтобы избежать ошибки при использовании Internet Explorer 6.
2. Удаляет лишние символы, такие как NULL и JavaScript-объекты Netscape 4.
3. Гарантирует, что HTML-объекты типа `&` правильно сформированы.
4. Гарантирует, что HTML-дескрипторы и их атрибуты правильно сформированы. В ходе этого процесса дескрипторы, которые не были включены в список разрешенных (во втором параметре `filter_xss()`), удаляются. Атрибут `style` также удаляется, потому что он может влиять на макет страницы, переопределяя CSS или скрывая контент за счет установки цвета спаммерской ссылки в цвет фона страницы. Любые атрибуты, начинающиеся с `on` (такие как `onclick` или `onfocus`), удаляются, потому что они представляют определения обработчиков событий JavaScript. Если вы знакомы с регулярными выражениями, вам будет интересно пошагово просмотреть функцию `filter_xss()` (в `modules/filter/filter.module`) и связанные с нею функции в отладчике.
5. Гарантирует, что никакие HTML-дескрипторы не содержат запрещенные протоколы. Разрешены следующие протоколы: `http`, `https`, `ftp`, `news`, `nnntp`, `telnet`, `mailto`, `irc`, `ssh`, `sftp` и `webcal`. Этот список можно изменить, устанавливая переменную `filter_allowed_protocols`. Например, добавление следующей строки в файл `settings.php` (см. комментарий о переопределении переменных в `settings.php`) ограничивает список только протоколами `http` и `https`:

```
$conf = array(
    'filter_allowed_protocols' => array('http', 'https')
);
```

Ниже приведен пример использования `filter_xss()` из модуля `modules/aggregator/aggregator.pages.inc`. Этот модуль имеет дело с потенциально опасными лентами RSS или Atom. Он готовит переменные к использованию следующим образом:

```
/**
 * Безопасно визуализирует HTML-контент в соответствии с разрешениями.
 *
 * @param $value
 *   Контент, который должен быть отфильтрован.
 * @return
 *   Отфильтрованный контент.
 */
function aggregator_filter_xss($value) {
    return filter_xss($value, preg_split('/\s+|<|>/',
        variable_get('aggregator_allowed_html_tags', '<a> <b> <br> <dd> <dl> <dt>
        <em> <i> <li> <ol> <p> <strong> <u> <ul>'), -1, PREG_SPLIT_NO_EMPTY));
}
```

Обратите внимание на вызов функции `aggregator_filter_xss()`, которая является оболочкой для функции `filter_xss()` и предоставляет массив разрешенных HTML-дескрипторов.

На заметку! В качестве упражнения, связанного с безопасностью, можно на примере любого имеющегося специального модуля проследить поступление введенной пользователем информации в систему, ее сохранение и вывод и удостовериться, что текст очищается где-то в ходе этого процесса.

Использование функции `filter_xss_admin()`

Иногда нужно, чтобы модуль генерировал HTML-код для административных страниц. Поскольку административные страницы должны быть защищены механизмом контроля доступа, предполагается, что пользователи, имеющие доступ к страницам администрирования, заслуживают большего доверия, чем обычные пользователи. Можно было бы определить специальный фильтр для административных страниц и использовать систему фильтров, но это решение оказалось бы слишком громоздким. Поэтому система предоставляет функцию `filter_xss_admin()`. Это — просто оболочка для `filter_xss()` с “либеральным” списком разрешенных дескрипторов, который включает в себя все дескрипторы, кроме `<script>`, `<object>` и `<style>`. Примером применения `filter_xss_admin()` может служить отображение предназначения сайта в теме:

```
if (drupal_is_front_page()) {
  $mission = filter_xss_admin(theme_get_setting('mission'));
}
```

Предназначение сайта может быть установлено только со страницы Configuration⇒“Site information” page (Конфигурация⇒Страница “Информация о сайте”), доступ к которой имеют только суперпользователь и пользователи с привилегией `administer site configuration` (администрирование конфигурации сайта) — как раз та ситуация, когда использование функции `filter_xss_admin()` оправдано.

Безопасная обработка URL

Часто модули принимают отправленные пользователями URL-адреса и отображают их. В этой ситуации необходим какой-нибудь механизм проверки того, что предоставленное пользователем значение является легитимным URL. Drupal предоставляет функцию `check_url()`, которая в действительности является лишь оболочкой для `filter_xss_bad_protocol()`. Она позволяет удостовериться, что протокол, указанный в URL, относится к числу разрешенных для применения на сайте Drupal (см. п. 5 в разделе “Использование функции `filter_xss()` для предотвращения межсайтовых атак с помощью сценариев”) и подвергает URL обработке функцией `check_plain()`.

Для проверки правильности формы URL служит функция `valid_url()`. Она проверяет синтаксис для URL типов `http`, `https` и `ftp` и производит проверку на предмет присутствия запрещенных символов; в случае успешной проверки `valid_url()` возвращает значение `TRUE`. Это — быстрый способ удостовериться, что пользователи не отправляют URL с протоколом `javascript`.

Внимание! То, что URL успешно прошел проверку синтаксиса, вовсе *не* означает, что он безопасен!

Если в URL передается некоторая информация — например, в строке запроса — с помощью функции `drupal_encode_path()` вместе с ним можно передать символы отмены. Функция `drupal_encode_path()` выполняет определенное кодирование символов косой черты (/) для совместимости с чистыми URL-адресами Drupal и затем вызывает PHP-функцию `rawurlencode()`. Функция `drupal_encode_path()` не является более безопасной, чем вызов `rawurlencode()` напрямую, но она удобна для создания кодированных строк, которые будут хорошо работать с Apache-модулем `mod_rewrite`.

Совет. Функция `drupal_encode_path()` представляет собой пример PHP-функции, помещенной в оболочку — `rawurlencode()` можно было вызвать непосредственно, но при этом не удалось бы воспользоваться автоматической обработкой несоответствий средой Drupal. Аналогичные функции-оболочки для работы со строками можно найти в `includes/unicode.inc`; например, вместо PHP-функции `strlen()` можно применять функцию `drupal_strlen()`.

Обеспечение безопасности запросов с помощью функции `db_query()`

Распространенный способ взлома веб-сайта — так называемое *внедрение SQL* (SQL injection). Исследуем модуль, написанный без учета соображений безопасности. Требуется реализовать простое перечисление заголовков всех нод определенного типа:

```
/*
 * Реализация hook_menu()
 */
function insecure_menu() {
  $items['insecure'] = array(
    'title' => 'Insecure Test',
    'page callback' => 'insecure_code',
    'access arguments' => array('access content'),
  );
  return $items;
}

/*
 * Обратный вызов меню, иницилируемый, когда пользователь
 * переходит на http://example.com/?q=insecure
 */
function insecure_code($type = 'story') {
  $output = "Searching for nodes of type: $type <br/>";

  $query = db_select('node', 'n');
  $query->fields('n', array('title'));
  $query->condition("n.type", $type);
  $result = $query->execute();

  $items = array();

  foreach($result as $row) {
    $items[] = $row->title;
  }

  if (sizeof($items) > 0) {
    $output .= theme('item_list', array('items' => $items));
  } else {
    $output .= "No nodes were found of type $type";
    // Ноды этого типа не найдены
  }
  return $output;
}
```

При переходе на `http://example.com/insecure` все работает так, как и ожидалось. Мы получаем SQL-код, а затем — список нод типа `story` (рис. 21.2).

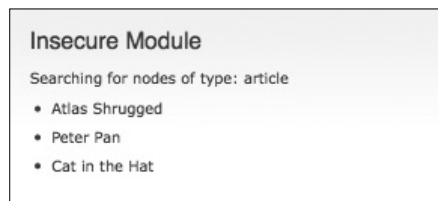


Рис. 21.2. Простой список заголовков нод типа `story`

Обратите внимание на передачу функции `insecure_code()` параметра `$type` со значением по умолчанию, равным `'story'`. Здесь использован тот факт, что система меню Drupal автоматически передает обратным вызовам дополнительные аргументы пути в виде параметров, в результате чего `http://example.com/insecure/page` возвратит все заголовки нод типа `'page'`, как показано на рис. 21.3.

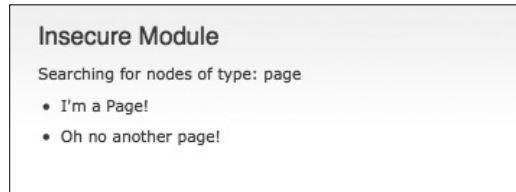


Рис. 21.3. Простой список заголовков нод типа `page`

Однако ситуацию можно еще улучшить. В данном случае URL должен содержать только члены из конечного набора — т.е. типы нод на сайте. Эти типы известны, поэтому всегда следует удостоверяться, что введенное пользователем значение присутствует в списке известных значений. Например, если на сайте разрешены только типы нод `page` и `article`, обработка ввода должна выполняться только в том случае, если именно эти типы переданы в URL. Добавим соответствующий код проверки:

```
function insecure_code($type = 'article') {
    $types = node_type_get_types();
    if (!isset($types[$type])) {
        watchdog('security', 'Possible SQL injection attempt!', array(),
            WATCHDOG_ALERT);
        return t('Unable to process request.');
```

```
    }

    $output = "Searching for nodes of type: $type <br/>";

    $query = db_select('node', 'n');
    $query->fields('n', array('title'));
    $query->condition("n.type", $type);
    $result = $query->execute();

    $items = array();

    foreach($result as $row) {
        $items[] = $row->title;
    }

    if (sizeof($items) > 0) {
        $output .= theme('item_list', array('items' => $items));
    } else {
        $output .= "No nodes were found of type $type";
    }
    return $output;
}
```

Здесь добавлена процедура проверки того, что `$type` является одним из известных типов нод, а также реализована запись полезного предупреждения для системных администраторов в случае неудачи этой проверки. Однако остаются и другие проблемы. Оператор SQL не делает различий между опубликованными и неопубликованными нодами, так что будут отображаться заголовки даже неопубликованных нод. Кроме того,

заголовки нод — это переданные пользователем данные, поэтому их нужно очищать перед выводом. Но в своем существующем виде код только получает заголовки из базы данных и отображает их. Давайте решим эти проблемы.

```
function insecure_code($type = 'article') {
    $types = node_type_get_types();
    if (!isset($types[$type])) {
        watchdog('security', 'Possible SQL injection attempt!', array(),
            WATCHDOG_ALERT);
        return t('Unable to process request.');
```

```
    }

    $output = "Searching for nodes of type: $type <br/>";

    $query = db_select('node', 'n');
    $query->fields('n', array('title'));
    $query->condition("n.type", $type);
    $query->condition("n.status", 1);
    $result = $query->execute();

    $items = array();

    foreach($result as $row) {
        $items[] = check_plain($row->title);
    }

    if (sizeof($items) > 0) {
        $output .= theme('item_list', array('items' => $items));
    } else {
        $output .= "No nodes were found of type $type";
    }

    return $output;
}
```

Теперь будут отображаться только неопубликованные ноды, а все заголовки перед отображением будут обрабатываться функцией `check_plain()`. Также удален отлаженный код. Этот модуль прошел долгий путь! Тем не менее, безопасность все еще под угрозой. Заметили, в чем проблема? Если нет, продолжайте читать дальше.

Сохранение конфиденциальности личных данных с помощью функции `hook_alter_query()`

Предыдущий пример вывода списка нод — довольно распространенная задача, решаемая дополнительными модулями (хотя теперь она менее актуальна, поскольку модуль `views` помогает легко определить списки нод через веб). Возникает вопрос: если на сайте активирован модуль контроля доступа к нодам, то где в предыдущем примере код, который гарантирует отображение пользователю только разрешенного ему подмножества нод? Совершенно верно: такой код отсутствует. Приведенный код отобразит все ноды заданного типа, *даже защищенные модулями доступа к нодам*. Он слишком “самонадеян”, чтобы интересоваться мнением других модулей! Давайте изменим его.

Код до изменения:

```
$query = db_select('node', 'n');
$query->fields('n', array('title'));
$query->condition("n.type", $type);
$query->condition("n.status", 1);
$result = $query->execute();
```

Код после изменения:

```
$query = db_select('node', 'n');
$query->fields('n', array('title'));
$query->condition("n.type", $type);
$query->condition("n.status", 1);
$query->addTag('node_access');
$result = $query->execute();
```

В параметр `$query` запроса добавлена инструкция `->addTag('node_access')`, которая вызывает функцию `hook_query_alter()` для изменения кода SQL в соответствии с ограничениями доступа, установленными разрешениями.

Динамические запросы

При наличии в коде SQL переменного количества значений, которые не могут быть определены до времени выполнения, необходимо использовать оператор `$query->condition(поле, массив_значений, 'IN')` для ограничения запроса динамическим списком значений, определенных вторым параметром. Ниже приведен пример применения этого подхода:

```
// $node_types — это массив, содержащий одно или более имен типов нод,
// таких как article, page, blog и т.д.
$node_types = array('article', 'page', 'blog');

// Подготовка и выполнение запроса с использованием списка типов нод.
$query = db_select('node', 'n');
$query->fields('n', array('title'));
$query->condition("n.type", $node_types, 'IN');
$query->condition("n.status", 1);
$query->addTag('node_access');
$result = $query->execute();
```

Разрешения и обратные вызовы страницы

Еще один аспект, который следует учитывать при написании собственных модулей — ключ `access arguments` (аргументы доступа) каждого элемента меню, определяемого в хуке меню. В приведенном ранее примере небезопасного кода применялись следующие аргументы доступа:

```
/*
 * Реализация hook_menu()
 */
function insecure_menu() {
  $items['insecure'] = array(
    'title' => 'Insecure Module',
    'description' => 'Example of how not to do things.',
    'page callback' => 'insecure_code',
    'access arguments' => array('access content'),
  );
  return $items;
}
```

Важно определить, кому разрешено получать доступ к этому обратному вызову. Разрешение типа `access content` (доступ к контенту) носит очень общий характер. Скорее всего, потребуется определить собственные разрешения с помощью функции `hook_permission()` и применять их для защиты обратных вызовов меню. Разрешения —

это уникальные строки, описывающие предоставляемые права доступа (подробнее они рассмотрены в разделе “Управление доступом” главы 4).

Поскольку реализация хука меню играет роль стража, который разрешает или запрещает пользователю доступ к охраняемому коду (через обратный вызов), особенно важно продумать предоставляемые разрешения.

Подделки межсайтовых запросов (CSRF)

Предположим, что вы зашли на сайт `drupal.org` и просматриваете его форумы. Затем вы отвлекаетесь на что-то другое и переходите на другой сайт. Некий злоумышленник на этом веб-сайте помещает дескриптор изображения подобный следующему:

```

```

При загрузке изображения веб-браузер запросит этот путь из сайта `drupal.org`. Поскольку в данный момент вы зарегистрированы на `drupal.org`, вместе с запросом браузер отправит cookie-набор. И здесь возникает вопрос: когда сайт `drupal.org` принимает запрос, признает ли он в вас зарегистрированного пользователя со всеми надлежащими правами доступа? Можно поспорить, что да! Эта ссылка на изображение, подsunутая злоумышленником, по сути, вынуждает пользователя щелкнуть на ссылке на `drupal.org`.

Первая защитная мера от такого рода атак состоит в том, чтобы никогда не использовать запросы GET для проведения каких-то изменений на сервере. В результате любой запрос, сгенерированный подобным образом, будет безвредным. API-интерфейс форм Drupal следует соглашению HTTP/1.1 о том, что метод GET не должен предпринимать никаких действий, кроме извлечения данных. Drupal использует метод POST исключительно для действий, связанных с изменениями на сервере (см. www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.1).

В качестве второй защитной меры API-интерфейс форм использует маркеры и уникальные идентификаторы для удостоверения в том, что данные формы, отправленные запросами POST, поступают от той же формы, которая была послана средой Drupal (подробнее об этом читайте в главе 11). При написании модулей используйте API-интерфейс форм — тогда этот вид защиты будет реализован автоматически. Любое действие, предпринимаемое модулем в результате ввода в форму, должно происходить *в функции отправки* формы. Это гарантирует защиту посредством API-интерфейса форм.

И, наконец, при необходимости запросы GET можно защищать также за счет использования маркера (сгенерированного функцией `drupal_get_token()`) в URL и проверки этого маркера с помощью функции `drupal_valid_token()`.

Безопасность файлов

Опасности, с которыми сталкивается среда Drupal при обработке файлов и файловых путей — те же, что и в других веб-приложениях.

Права доступа к файлу

Права доступа должны устанавливаться таким образом, чтобы пользователь не мог манипулировать (добавлять, переименовывать или удалять) файлами. Веб-сервер должен иметь доступ к файлам и каталогам Drupal только для чтения. Исключением являются пути файловой системы. Ясно, что веб-сервер должен иметь доступ к этим каталогам, чтобы получить возможность записывать загруженные на него файлы.

Защищенные файлы

Файл `.htaccess`, поставляемый вместе с Drupal, содержит в себе следующие строки:

```
# Protect files and directories from prying eyes.
<FilesMatch "\.(engine|inc|info|install|make|module|profile|test|po|sh|?
.*sql|theme|tpl(\.php)?|xhtml)$|^(\.?.*|Entries.*|Repository|Root|Tag|Template)$">
  Order allow,deny</FilesMatch>
# Protect files and directories from prying eyes.
<FilesMatch "\.(engine|inc|info|install|module|profile|po
|sh|.*sql|theme|tpl(\.php)?|xhtml)$|^(\code-style\.pl
|Entries.*|Repository|Root|Tag|Template)$">Order allow,deny</FilesMatch>
```

Директива `Order` установлена в `allow,deny`, но никаких директив `Allow` или `Deny` не включено. Это означает, что подразумеваемым поведением является запрет (`deny`). Иными словами, все запросы файлов, перечисленных в табл. 21.2, отклоняются.

Таблица 21.2. Файлы, отклоняемые регулярным выражением директивы `FilesMatch` в файле `.htaccess` Drupal

Файлы	Описание
Файлы с окончаниями	
<code>.engine</code>	Механизмы шаблона
<code>.inc</code>	Библиотечные файлы
<code>.info</code>	Файлы <code>.info</code> модулей и тем
<code>.install</code>	Файлы <code>.install</code> модулей
<code>.module</code>	Файлы модулей
<code>.make</code>	Сборочные файлы (make-файлы)
<code>.profile</code>	Профили инсталляции
<code>.po</code>	Файлы переносимых объектов (переводы)
<code>.sh</code>	Сценарии оболочек
<code>.*sql</code>	SQL-файлы
<code>.test</code>	Тестовые сценарии
<code>.theme</code>	Темы PHP
<code>.tpl.php</code>	Файлы шаблонов PHPTemplate
<code>.tpl.php4</code>	Файлы шаблонов PHPTemplate
<code>.tpl.php5</code>	Файлы шаблонов PHPTemplate
<code>.xhtml</code>	Файлы XHTML
Файлы начинающиеся с	
<code>Entries</code>	Файл CVS
Файлы с именем	
<code>Repository</code>	Файл CVS
<code>Root</code>	Файл CVS
<code>Tag</code>	Файл CVS
<code>Template</code>	Файл CVS

Загрузка файлов на сервер

Если модулю разрешено выполнять загрузку файлов на сервер, эти файлы должны помещаться в специальный каталог, и доступ должен обеспечиваться кодом.

Если загрузка файлов разрешена, и на странице Configuration⇒File system (Конфигурация⇒Файловая система) выбран приватный каталог загрузки, на этой же странице общий доступ к пути файловой системы должен быть полностью запрещен.

Имена файлов и пути

Никакой информации об именах файлов и путях, поступившей от пользователя, доверять нельзя! При написании модуля в расчете на то, что код должен получать некий файл `somefile.txt`, следует помнить, что вместо него может быть получено нечто иное, например:

```
../somefile.txt           // Файл в родительском каталоге.
../settings.php          // Целевой файл.
somefile.txt; cp ../settings.php ../settings.txt // Попытка выполнения
                                                    // команды оболочки.
```

Первые два примера пытаются манипулировать путем файла, включая две точки, означающие родительский каталог операционной системы сервера. Последний пример предполагает выполнение программистом команды оболочки, и включает точку с запятой, в результате чего после выполнения этой команды будет выполнена дополнительная команда, которая сделает `settings.php` читаемым и тем самым откроет имя и пароль пользователя базы данных. Все приведенные примеры исходят из того, что права доступа к файлам установлены некорректно, и что веб-сервер в действительности имеет доступ для записи к каталогам вне пути файловой системы веб-сервера.

Всякий раз, используя пути файлов, следует вызывать функцию `file_valid_uri()`, как показано ниже:

```
if (!file_valid_uri($uri) {
    // Прервать! Указан непредусмотренный идентификатор URI файла!
}
```

Функция `file_valid_uri()` выяснит, содержит ли URI допустимую схему для операций с файлами. В общем случае вряд ли стоит в качестве первого проекта Drupal братья за что-нибудь вроде еще одного великолепного модуля управления файлами. Вместо этого лучше изучите доступные модули работы с файлами, которые существуют достаточно долгое время.

Кодирование заголовков почты

При написании любого кода, который принимает введенную пользователем информацию и встраивает ее в сообщение электронной почты, необходимо учитывать следующие два факта.

- Заголовки сообщений электронной почты отделены символами перевода строки (только символы перевода строки, за которыми не следуют символы пробела или табуляции, трактуются в качестве разделителей заголовка).
- Если проверка того, что вводимая пользователями информация не содержит символы перевода строки, отсутствует, пользователи могут внедрить собственные заголовки в тело почтового сообщения.

Например, предположим, ожидается, что пользователь введет тему своего сообщения, а он вводит строку, в которую вставлены символы перевода строки (%0A) и пробела (%20):

```
Have a nice day%0ABcc:spamtargеt@example.com%0A%0AL0w%20c0st%20mortgage!
```

Результат был бы следующим:

```
Subject: Have a nice day
Bcc: spamtargеt@example.com

L0w c0st mortgage!
...
```

По этой причине встроенная почтовая функция Drupal по имени `drupal_mail()` в `includes/mail.inc` подвергает все заголовки обработке функцией `mime_header_encode()` с целью их очистки. Любые непечатные символы будут преобразованы в пригодные для печати символы стандарта ASCII согласно документу RFC 2047 и таким образом нейтрализованы. Это преобразование подразумевает предварение символа префиксом `=?UTF-8?B?` и последующую печать символа, закодированного алгоритмом Base64, с добавленным символов `?`.

Мы настоятельно рекомендуем применять функцию `drupal_mail()`; в противном случае вызовы `mime_header_encode()` придется выполнять самостоятельно.

Файлы для рабочей среды

Не все файлы, включенные в дистрибутив Drupal, необходимы для рабочих сайтов. Например, если сделать `CHANGELOG.txt` доступным на рабочем сайте, любой веб-пользователь сможет выяснить применяемую версию Drupal (разумеется, хакеры располагают и другими способами обнаружения того, что используется Drupal; см. <http://lullabot.com/articles/is-site-running-drupal>). В табл. 21.3 перечислены файлы и/или каталоги, необходимые для функционирования среды Drupal после ее инсталляции; все прочие могут быть удалены из рабочего сайта (с обязательным сохранением копий). Другой способ защиты — запрещение доступа к веб-серверу для чтения.

Таблица 21.3. Файлы и каталоги, необходимые для функционирования Drupal

Файл/каталог	Назначение
<code>.htaccess</code>	Безопасность, чистые URL и поддержка кэширования на Apache
<code>cron.php</code>	Позволяет запускать регулярно планируемые задачи
<code>includes/</code>	Библиотеки функций
<code>index.php</code>	Главная точка входа для запросов Drupal
<code>misc/</code>	JavaScript и графика
<code>modules/</code>	Модули ядра
<code>robots.txt</code>	Предотвращает взлом сайта известными роботами
<code>sites/</code>	Специфичные для сайта модули, темы и файлы
<code>themes/</code>	Основные темы
<code>xmlrpc.php</code>	Конечная точка XML-RPC; необходима, только если сайт будет принимать входящие запросы XML-RPC
<code>authorize.php</code>	Административный сценарий для выполнения авторизованных операций с файлами

Поддержка SSL

По умолчанию Drupal обрабатывает информацию, предоставляемую пользователями при входе в сайт, в виде простого текста по протоколу HTTP. Однако Drupal поддерживает и протокол HTTPS, если только сам веб-сервер поддерживает его. Никакой модификации Drupal для этого не требуется.

Автономный PHP

Иногда может возникнуть необходимость в создании автономного файла `.php` вместо того, чтобы помещать соответствующий код в модуль Drupal. При этом следует учитывать соображения безопасности.

На заметку! Следующий код предназначен для учебных целей. Лучший подход предусматривает использование возможностей, предлагаемых Drush (<http://drush.ws>) — оболочкой командной строки и интерфейса написания сценариев для Drupal. Оболочка Drush автоматически выполняет полную начальную загрузку Drupal, прежде чем выполнять сценарий. Это избавляет от необходимости включать код начальной загрузки в PHP-файл, что минимизирует риски безопасности, отраженные в следующем примере кода.

Предположим, что для тестирования своего веб-сайта вы наспех написали некоторый сомнительный код, стремясь вставить пользователей в базу данных, чтобы оценить производительность сайта при обслуживании множества пользователей. Допустим, вы назвали его `testing.php` и поместили в корень сайта Drupal, вслед за `index.php`. Затем вы помечаете его закладкой в браузере и всякий раз, когда требуется обновить таблицу пользователей, выбираете эту закладку.

```
<?php
/**
 * Этот сценарий генерирует пользователей в целях тестирования.
 */
// Эти строки — все, что требуется для получения полного
// доступа к функциональным возможностям Drupal.

include_once 'includes/bootstrap.inc';
drupal_bootstrap(DRUPAL_BOOTSTRAP_FULL);

db_delete('users')
  ->condition('uid', '1', '>')
  ->execute();

for ($i = 2; $i <= 5000; $i++) {
  $name = $i;
  $pass = md5(user_password());
  $mail = $name .'@localhost';
  $status = 1;
  db_insert('users')
    ->fields(array('name' => $name, 'pass' => $pass, 'mail' => $mail,
      'status' => $status, 'created' => time(), 'access' => time()),
    ->execute();
}

print t('Users have been created.');// Пользователи созданы
```

Это полезно для тестирования, но вообразите, что произошло бы, если бы вы забыли о существовании этого сценария, и он выполнил свою работу в рамках рабочего сайта!

Любой, кому удалось выяснить URL вашего сайта `http://example.com/testing.php` смог бы удалить пользователей посредством единственного запроса. Поэтому важно, чтобы даже наспех созданные сценарии включали в себя проверку безопасности, как показано в следующем примере:

```
<?php
/**
 * Этот сценарий генерирует пользователей в целях тестирования.
 */
// Эти строки — все что требуется для получения полного
// доступа к функциональным возможностям Drupal.

include_once 'includes/bootstrap.inc';
drupal_bootstrap(DRUPAL_BOOTSTRAP_FULL);

// Проверка безопасности; выполнение сценария разрешено только
администратору сайта.
global $user;
if ($user->uid != 1) {
    print t('Not authorized.');
```

```
    exit();
}

db_delete('users')
    ->condition('uid', '1', '>')
    ->execute();

for ($i = 2; $i <= 10; $i++) {
    $name = $i;
    $pass = md5(user_password());
    $mail = $name . '@localhost';
    $status = 1;

    db_insert('users')
        ->fields(array('name' => $name, 'pass' => $pass, 'mail' => $mail,
            'status' => $status, 'created' => time(), 'access' => time())),
        ->execute();
}

print t('Users have been created.');
```

Из сказанного следуют два важных практических вывода.

- Даже в наспех созданных сценариях предусматривайте проверку безопасности, предпочтительно запускаемую из шаблона, который включает нужный код.
- Помните, что важной частью развертывания является удаление или отключение кода тестирования.

Безопасность AJAX и атаки повторением запроса

Важный, заслуживающий внимания аспект безопасности, связанный с такими возможностями AJAX, как jQuery — хотя обычно серверный код AJAX разрабатывается исходя из предположения, что он будет вызываться из JavaScript, ничто не мешает злонамеренному пользователю выполнять вызовы AJAX непосредственно (например, из утилит командной строки, таких как `curl` или `wget`, или даже просто вводя URL в веб-браузере). Удостоверьтесь, что код протестирован с этих точек зрения.

Безопасность API-интерфейса форм

Одна из выгод использования API-интерфейса форм состоит в том, что большая часть мер безопасности уже реализована. Например, Drupal проверяет, что значение, которое пользователь выбрал в раскрывающемся списке, соответствует одному из значений, предлагаемых Drupal. API-интерфейс форм использует установленную последовательность событий, таких как построение формы, проверка ее достоверности и запуск. Данные, введенные пользователем, не следует использовать до выполнения фазы проверки достоверности, поскольку тогда они не будут проверены. Например, при использовании значения из массива `$_POST` нет никакой гарантии, что пользователь не изменил его. Кроме того, во всех случаях, когда это возможно, для передачи информации вместе с формой применяйте элемент `#value`, а не скрытые поля, поскольку злоумышленники могут манипулировать скрытыми полями, но не имеют доступа к элементам `#value`.

Любые отправленные пользователем данные, применяемые для построения формы, должны быть надлежащим образом очищены подобно любому другим отправленным пользователем данным, как показано в следующем примере.

Этот код не является безопасным:

```
$form['foo'] = array(
  '#type' => 'textfield',
  '#title' => $node->title,                // Уязвимость XSS!
  '#description' => 'Teaser is: ' . $node->teaser, // Уязвимость XSS!
  '#default_value' => check_plain($node->title), // Излишне.
);
```

А это безопасный код:

```
$form['foo'] = array(
  '#type' => 'textfield',
  '#title' => check_plain($node->title),
  '#description' => t('Teaser is: @teaser', array('@teaser' => $node->teaser)),
  '#default_value' => $node->title,
);
```

Нет необходимости подвергать значение по умолчанию обработке функцией `check_plain()`, поскольку функция темы для типа элемента формы (в данном случае — `theme_textfield()` из `includes/form.inc`) уже делает это.

Внимание! При написании собственных функций темы или переопределении функции темы по умолчанию Drupal всегда задавайтесь вопросом, очищен ли любой ввод пользователя, и дублируйте эту меру безопасности в своем коде.

Более подробно API-интерфейс форм рассматривался в главе 11.

Защита учетной записи суперпользователя

Вероятно, простейший способ получить информацию доступа к веб-сайту Drupal — позвонить наивной секретарше и сказать: “Привет, это Джо. <Здесь можно добавить несколько комплиментов.> Я из команды компьютерной поддержки, и у нас тут возникли небольшие проблемы с веб-сайтом. С каким именем и паролем вы обычно входите на сайт?” К сожалению, многие люди спокойно предоставляют эту информацию в ответ на подобный запрос. Хотя технология важна, но обучение пользователей — лучшая защита от атак подобного рода.

Вот почему рекомендуется никогда никому не назначать права пользователя с идентификатором 1 (суперпользователь). Вместо этого каждое лицо, которое будет поддерживать веб-сайт, должно получать лишь столько полномочий, сколько необходимо для выполнения им своих обязанностей. Это позволит минимизировать ущерб в случае возникновения бреши в системе безопасности.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Ни при каких обстоятельствах не доверять данным, введенным пользователями.
- Уметь преобразовывать введенные пользователем данные, чтобы сделать их безопасными для отображения.
- Знать, каким образом избегать XSS-атак.
- Знать, как избегать атак внедрением SQL.
- Уметь написать код, который учитывает требования модулей доступа к нодам.
- Знать, как избегать атак CSRF.
- Знать, каким образом Drupal защищает загруженные на сервер файлы.
- Знать, как избегать внедрения опасного кода в заголовки почтовых сообщений.

ГЛАВА 22

Передовые приемы разработки

В этой главе вы найдете множество кратких советов по кодированию и наиболее рациональным приемам, которые превратят вас в полноправного гражданина сообщества Drupal и позволят в будущем лишь изредка прибегать к помощи клавиатуры. Глава начинается с представления стандартов кодирования Drupal, после чего будет показано, как создавать документацию, которая поможет другим разработчикам понять ваш код. Вы научитесь быстро находить нужное в кодовой базе Drupal, кратко ознакомитесь с управлением версиями, с процедурами сопровождения модуля, а также с отладкой и профилированием кода.

Стандарты кодирования

Сообщество Drupal пришло к соглашению, что для повышения удобочитаемости и облегчения ее освоения новыми разработчиками кодовая база системы Drupal должна иметь стандартизованный внешний вид и поведение. Разработчиков дополнительных модулей призывают также придерживаться этих стандартов. Если уж быть полностью откровенным, ваши модули не будут восприниматься всерьез, если вы не будете придерживаться стандартов кодирования. Сначала будут описаны эти стандарты, а затем представлено несколько автоматизированных инструментов, которые облегчают проверку написанного кода (и даже его корректирование должным образом).

Выравнивание строк и пробелы

Для выравнивания кода в Drupal используются два пробела, а не символы табуляции. В большинстве редакторов есть возможность устанавливать автоматическую замену символов табуляции пробелами. Поэтому в них по-прежнему можно использовать клавишу <Tab> для выравнивания, если это стало привычкой. Строки не должны завершаться пробелами.

В файлах в качестве символа конца строки должен использовать символ `\n` из Unix, а не стандартный символ конца строки `\r\n` из Windows. Все текстовые файлы должны заканчиваться одиночным символом новой строки (`\n`).

Операции

Все символы бинарных операций, такие как `+`, `-`, `=`, `!=`, `==`, `>` и т.д., должны помещаться между пробелами. Например, операция присваивания должна быть сформати-

рована как `c = a + b`, а не `c=a+b`. Унарные операции, такие как `++`, не должны содержать пробел между символом операции и переменной, к которой она применяется.

Приведение типов

В операции приведения типа между *(тип)* и *\$переменная* должен быть помещен пробел, как в `(int) $count`.

Управляющие структуры

Управляющие структуры, такие как `if`, `for`, `while` и `switch`, должны содержать один пробел между управляющим ключевым словом и открывающей круглой скобкой, чтобы их можно было отличить от вызовов функций. Например, в следующем операторе `if` демонстрируется правильное применение и размещение открывающей круглой скобки:

```
if (условие1 || условие2) {
    какое-то действие;
}
elseif (условие3 && условие4) {
    другое действие;
}
else {
    еще одно действие;
}
```

Настоятельно рекомендуется применять фигурные скобки даже в тех ситуациях, когда они формально не обязательны. Это способствует повышению читабельности кода и снижению вероятности логических ошибок при добавлении новых строк.

Операторы переключения (`switch`) форматируются следующим образом:

```
switch (условие) {
    case 1:
        действие1;
        break;
    case 2:
        действие2;
        break;
    default:
        действие_по_умолчанию;
}
```

Формат операторов `do-while` имеет следующий вид:

```
do {
    действия;
} while ($условие);
```

Вызовы функций

В вызовах функций должно быть по одному пробелу до и после операций (`=`, `<`, `>` и т.д.) и не должно быть пробела между именем функции и открывающей круглой скобкой. Также не должно быть пробела между открывающей круглой скобкой функции и ее первым параметром. Средние параметры функции разделяются запятой и пробелом, а последний параметр не должен содержать пробела между собой и закрывающей круглой скобкой. Следующие примеры иллюстрируют эти положения.

Неправильно

```
$var=foo ($bar,$baz);
```

Правильно

```
$var = foo($bar, $baz);
```

Из этого правила существует одно исключение. Блоки связанных присваиваний могут содержать дополнительные пробелы между операциями присваивания, если это способствует удобочитаемости:

```
$a_value      = foo($b);
$another_value = bar();
$third_value   = baz();
```

Объявления функций

Между именем функции и открывающей скобкой не должно быть пробела. При написании функций, использующих для некоторых параметров значения по умолчанию, указывайте такие параметры последними. Кроме того, если функция генерирует любые полезные данные, рекомендуется возвращать эти данные вызывающей функции на тот случай, если они ей понадобятся. Ниже приведены некоторые примеры объявлений функций:

Неправильно

```
function foo ($bar = 'baz', $qux){
    $value = $qux + some_function($bar);
}
```

Правильно

```
function foo($qux, $bar = 'baz') {
    $value = $qux + some_function($bar);
    return $value;
}
```

Имена функций

Имена функций в Drupal записываются в нижнем регистре и основаны на имени модуля или системы, к которым они относятся. Это соглашение позволяет избежать конфликтов пространств имен. Для отделения описательной части имени функции используются символы подчеркивания. После имени модуля должно следовать имя функции, состоящее из глагола и существительного — объекта этого глагола, например, `modulename_verb_object()`. В первом из приведенных ниже примеров неправильно именованная функция не имеет префикса — имени модуля, а объект и глагол переставлены местами. В последующем примере эта ошибка устранена.

Неправильно

```
function some_text_munge() {
    ...
}
```

Правильно

```
function mymodule_munge_some_text() {
    ...
}
```

Приватные функции следуют тем же соглашениям, что и все прочие, но снабжены префиксом в виде символа подчеркивания (`_`).

Вызовы конструктора класса

При вызове конструкторов классов без параметров всегда добавляйте круглые скобки, как показано в следующем примере:

```
$foo = new MyClassName();
```

Это обеспечивает единообразный вид с конструкторами, имеющими аргументы:

```
$foo = new MyClassName($arg1, $arg2);
```

Обратите внимание, что если имя класса — переменная, вначале будет вычислена переменная для получения имени класса, и лишь затем конструктор будет вызван. Следующий пример иллюстрирует использование переменной в качестве имени класса:

```
$bar = 'MyClassName';
$foo = new $bar();
$foo = new $bar($arg1, $arg2);
```

Массивы

Массивы форматируются с пробелами, разделяющими каждый элемент и каждую операцию присваивания. Если блок массива занимает больше 80 символов, каждый элемент должен быть помещен в собственную строку. Рекомендуется всегда помещать каждый элемент в собственную строку; такой код проще читать и сопровождать. Это облегчает также добавление или удаление элементов массива.

Неправильно

```
$fruit['basket'] = array('apple'=>TRUE, 'orange'=>FALSE, 'banana'=>TRUE,
    'peach'=>FALSE);
```

Правильно

```
$fruit['basket'] = array(
    'apple' => TRUE,
    'orange' => FALSE,
    'banana' => TRUE,
    'peach' => FALSE,
);
```

На заметку! Запятая в конце последнего элемента массива не считается ошибкой, и PHP допускает такой синтаксис. Это может даже предотвратить ошибку в случае, если разработчик решит добавить или удалить элемент в конце списка массива. Описанное соглашение является допустимым и рекомендуемым, но не обязательным.

Создавая внутренние массивы Drupal, такие как элементы меню или определения форм, всегда указывайте только один элемент в каждой строке:

```
$form['flavors'] = array(
    '#type'          => 'select',
    '#title'         => t('Flavors'),
    '#description'   => t('Choose a flavor.'),
    '#options'       => $flavors,
);
```

Кавычки

В Drupal отсутствует жесткий стандарт в отношении использования одинарных либо двойных кавычек в тех или иных случаях. Когда возможно, сохраняйте единооб-

разие внутри каждого модуля и учитывайте персональные стили других разработчиков. С учетом сказанного помните о следующем нюансе: одинарные кавычки работают быстрее, поскольку синтаксическому анализатору не приходится искать встроенные переменные. Одинарные кавычки рекомендуется применять во всех ситуациях, кроме перечисленных ниже.

1. При использовании встроенных переменных, например, "<h2>\$header</h2>".
2. В переведенных строках, в которых можно избежать необходимости отключения одинарных кавычек, заключая строку в двойные кавычки. Пример такой строки — "He's a good person.". С применением одинарных кавычек она бы выглядела как 'He\'s a good person.'. Подобное отключение интерпретации символов может неправильно обрабатываться генераторами текстового перевода файла .pot. Кроме того, эта форма представления текста несколько неудобна для чтения.

Конкатенация строк

Для повышения удобочитаемости между символом точки и объединяемыми фрагментами следует всегда вставлять пробел, как показано в следующем примере:

```
$string = 'Foo' . $bar;
$string = $bar . 'Foo';
$string = bar() . 'Foo';
$string = 'foo' . 'bar';
```

При конкатенации простых переменных можно использовать двойные кавычки, помещая переменную внутрь них:

```
$string = "Foo $bar";
```

Комментарии

Система Drupal следует большинству рекомендаций относительно стиля комментариев Doxygen. Все блоки документации должны использовать следующий синтаксис:

```
/**
 * Документация.
 */
```

Начальные пробелы перед звездочками (*) в строках, следующих за первой, обязательны.

На заметку! Doxygen — это дружелюбный по отношению к PHP генератор документации. Он извлекает PHP-комментарии из кода и генерирует удобную для чтения документацию. Дополнительную информацию можно найти на сайте <http://www.doxygen.org>.

При документировании функции блок документации должен непосредственно предшествовать документируемой функции без промежуточных пустых строк.

Drupal воспринимает перечисленные ниже конструкции Doxygen. Хотя в списке перечислено большинство из таких конструкций, обязательно обратитесь к сайту Doxygen за дополнительной информацией по их использованию.

- @mainpage
- @file
- @defgroup
- @ingroup

- @addtogroup (как синоним @ingroup)
- @param
- @return
- @link
- @see
- @{
- @}

Преимущество соблюдения этих стандартов состоит в том, что можно автоматически генерировать документацию для разрабатываемых модулей, используя дополнительный модуль API-интерфейса. Модуль API-интерфейса — это реализация подмножества спецификации генератора документации Doxygen, настроенная для создания вывода, который максимально соответствует кодовой базе Drupal. Посмотреть этот модуль в действии можно на сайте <http://api.drupal.org>, а получить дополнительные сведения о модуле API-интерфейса — по адресу <http://drupal.org/project/api>.

Примеры документирования

Рассмотрим структуру модуля сверху донизу и попутно остановимся на различных типах документации.

Прежде чем объявлять функции, уделите время документированию выполняемых модулем действий, используя следующий формат:

```
/**
 * @file
 * Однострочное описание/резюме выполняемых модулем действий.
 *
 * Один-два абзаца с расширенным описанием модуля и его поведения.
 */
```

Документирование констант

Константы PHP должны записываться прописными буквами, с символами подчеркивания, разделяющими отдельные слова. При определении PHP-констант неплохо приводить объяснения их предполагаемого назначения, как показано в следующем фрагменте кода:

```
/**
 * Идентификатор ролей аутентифицированных пользователей; должен совпадать
 * с идентификатором, записанным в таблице "role".
 */
define('DRUPAL_AUTHENTICATED_RID', 2);
```

Документирование функций

Документация по функции должна использовать следующий синтаксис:

```
/**
 * Краткое описание, начинающееся с глагола.
 *
 * Далее следует подробное описание.
 *
 * @param $foo
 *   Описание переменной $foo.
 * @param $bar
```

430 Глава 22. Передовые приемы разработки

```
* Описание переменной $bar.  
* @return  
* Описание информации, возвращаемой функцией.  
*/  
function name_of_function($foo, $bar) {  
    ...  
    return $baz;  
}
```

Краткое описание должно начинаться с глагола. Взглянем на пример из ядра Drupal, который можно найти в `system.module`:

```
/**  
 * Добавить в форму кнопки по умолчанию и установить префикс.  
 *  
 * @param $form  
 * Ассоциативный массив, содержащий структуру формы.  
 *  
 * @return  
 * Структура формы.  
 *  
 * @see system_settings_form_submit()  
 * @ingroup forms  
 */  
function system_settings_form($form) {  
    ...  
}
```

Приведенный пример содержит несколько новых конструкций Doxygen.

- `@see` указывает ссылки на другие функции. Приведенный код представляет собой определение формы, поэтому `@see` указывает на обработчик отправки формы. Когда модуль API-интерфейса анализирует этот код для генерации документации (вроде той, которая доступна на сайте <http://api.drupal.org>), он преобразует имя функции, следующее за `@see`, в ссылку, реагирующую на щелчки.
- `@ingroup` связывает вместе набор соответствующих функций. В рассмотренном примере она создает группу функций, которые предоставляют определения форм. Можно указывать любой имя группы. Возможные имена, пригодные для использования ядром: `batch`, `database`, `file`, `format`, `forms`, `hooks`, `image`, `menu`, `node_access`, `node_content`, `schemaapi`, `search`, `themeable` и `validation`.

Совет. Все функции определенной группы можно просмотреть на сайте <http://api.drupal.org>. Например, функции построителя форм перечислены на <http://api.drupal.org/api/group/forms/7>, а функции, допускающие темизацию — на <http://api.drupal.org/api/5/group/themeable>.

Функции, реализующие общие конструкции Drupal, такие как хуки или функции проверки/отправки форм, могут опускать полный синтаксис `@param` и `@return`, но должны содержать однострочное описание того, что они делают, как показано в следующем примере:

```
/**  
 * Проверить достоверность формы настроек книги.  
 *  
 * @see book_admin_settings()  
 */
```

```
function book_admin_settings_validate($form, &$form_state) {
    ...
}
```

Полезно знать, что функция является функцией обратного вызова меню (т.е. преобразуется в URL с помощью функции `hook_menu()`):

```
/**
 * Обратный вызов меню; печатает список всех книг.
 */
function book_render() {
    ...
}
```

Документирование реализаций хуков

Когда функция представляет собой реализацию хука, нет необходимости документировать сам хук. Достаточно указать, какой конкретно хук реализует функция, например:

```
/**
 * Реализация hook_theme().
 */
function statistics_theme() {
    ...
}
```

Включение кода

При каждом безусловном включении файла класса используйте функцию `required_once()`. При каждом включении файла класса применяйте функцию `include_once()`. Любая из них гарантирует только однократное включение файлов классов. Они совместно используют один и тот же список файлов, поэтому можно не беспокоиться об их сочетании. Файл, включенный функцией `require_once()`, не будет включаться снова вызовом функции `include_once()`. Ниже приведен пример использования функции `require_once`:

```
require_once(DRUPAL_ROOT . '/' . variable_get('cache_inc', 'includes/cache.inc'));
```

Дескрипторы PHP-кода

Для обозначения PHP-кода всегда используйте дескриптор `<?php ?>`, а не его сокращенную форму `<? ?>`. Это требуется для обеспечения совместимости с Drupal, а также служит наиболее переносимым способом включения PHP-кода в различных операционных системах. Сочетание символов `>?>` всегда опускается в конце файла кода; это относится как к включаемым модулям, так и к включаемым файлам. Это обусловлено следующими причинами.

1. Стремление к исключению возможности появления нежелательных пробелов в конце файлов, которые могут приводить к ошибкам “заголовок уже отправлен”, проблемам проверки допустимости кода XHTML/XML и другим проблемам.
2. Закрывающий символ-разделитель является необязательным.
3. PHP.net автоматически удаляет закрывающий разделитель из конца своего файла, тем самым определяя наиболее рациональный подход.

Однако закрывающий дескриптор `>?>` следует использовать при смешанном применении кода PHP и HTML, когда HTML-код следует за PHP-кодом.

Символы точки с запятой

Язык PHP требует использования символов точки с запятой в конце большинства строк, но позволяет опускать их в конце блоков кода. В то же время стандарты кодирования Drupal требуют их применения даже в конце блоков кода.

URL-адреса примеров

Для всех URL-адресов примеров из RFC 2606 используйте `example.com`.

Соглашения по именованию

Имена функций и переменных должны вводиться строчными буквами, а отдельные слова должны разделяться символами подчеркивания. Кроме того, во избежание конфликта между модулями, имена функций должны предваряться префиксом группирования/имени модуля.

Имена постоянных переменных (переменных/настроек, определенных посредством Drupal-функций `variable_get()`/`variable_set()`), должны вводиться строчными буквами, а слова должны разделяться символами подчеркивания. Чтобы избежать конфликта между модулями, имена должны предваряться префиксом группирования/имени модуля.

Константы должны записываться прописными буквами, с символами подчеркивания, разделяющими отдельные слова. Это относится также к предопределенным константам PHP, таким как `TRUE`, `FALSE` и `NULL`. Имена констант, определенных в модулях, должны предваряться префиксом имени соответствующего модуля, записанного прописными буквами.

Имена глобальных констант должны начинаться с символа подчеркивания, за которым следует имя модуля/темы, записанное строчными буквами, и еще один символ подчеркивания.

Классы должны именоваться в стиле Pascal, например, `DatabaseConnection`. Имена методов и свойств классов должны использовать “верблюжий” стиль наподобие `$lastStatement`. Применения приватных методов и свойств классов следует избегать. Классы нужно определять как защищенные (`protected`), чтобы при необходимости другой класс мог расширять данный класс и изменять методы. Защищенные и общедоступные (`public`) методы и свойства не должны использовать префикс символа подчеркивания.

Для упрощения просмотра в системах Windows расширением имен всех файлов документации должно быть `.txt`. Кроме того, имена таких файлов должны записываться всеми прописными буквами, а их расширения — строчными (например, `README.txt`).

Программная проверка стиля кодирования

Существуют два основных подхода к проверке соответствия своего стиля кодирования стандартам кодирования Drupal: один использует сценарий Perl, а другой — дополнительный модуль.

Использование сценария `code-style.pl`

Каталог `scripts` корневого каталога Drupal содержит Perl-сценарий `code-style.pl`, который проверяет стиль кодирования для Drupal. Ниже описано, как его использовать.

Сначала измените разрешения, чтобы сделать файл исполняемым — в противном случае будет генерироваться ошибка `Permission denied` (Доступ запрещен). Это можно сделать из командной строки с помощью `chmod`:

```
$ cd scripts
$ ls -l | grep code-style
-rw-r--r-- 1 jvandyk jvandyk 4946 Feb 15 2007 code-style.pl

$ chmod u+x code-style.pl
$ ls -l | grep code-style
-rwxr--r-- 1 jvandyk jvandyk 4946 Feb 15 2007 code-style.pl
```

Пользователям Windows не нужно волноваться об изменении разрешений файла, но перед запуском `code-style.pl` им, возможно, придется удостовериться в том, что Perl установлен. Информация о Perl доступна по адресу www.perl.org.

Теперь `code-style.pl` можно запустить, перейдя в каталог, где расположен оцениваемый модуль или файл. Ниже показано, как может выглядеть команда запуска:

```
$ ./code-style.pl ../modules/node/node.module
```

Обычно вывод программы будет представлен в следующем формате:

```
номер строки : 'ошибка' -> 'исправление' : содержимое строки
```

Например, следующий сценарий сообщает, что в строке 30 модуля `foo.module`, которая в данный момент содержит код `$a=1`; требуются пробелы до и после операции присваивания.

```
foo.module30: '=' -> ' = ' : $a=1;
```

На заметку! Внимательно относитесь к положительным результатам оценки. Хотя сценарий `code-style.pl` довольно хорошо выполняет свою работу, он не совершенен, и каждый его отчет требует вдумчивого анализа.

Использование модуля кодировщика

По адресу <http://drupal.org/project/coder> доступно подлинное сокровище, которое сэкономит массу времени и усилий. Это модуль `coder` — модуль, который проверяет код других модулей.

Чтобы заставить модуль `coder` просмотреть данный модуль, щелкните на появившейся ссылке `Code review` (Просмотр кода) на странице навигации сайта и выберите необходимый способ просмотра, а также модуль или тему, которую требуется проверить. Или же воспользуйтесь удобной ссылкой `Code Review` (Просмотр кода), которую этот модуль предоставляет в списке модулей.

Совет. Использование модуля `coder` следует считать обязательным, если вы всерьез намерены придерживаться соглашений по кодированию для Drupal.

Можно даже пойти дальше и использовать сценарий `coder_format.php`, который поставляется вместе с модулем `coder`. Этот сценарий действительно исправляет ошибки форматирования кода. Вот как `coder_format.php` проверяет модуль `annotate`, который был создан в главе 2:

```
$ cd sites/all/modules
$ php contrib/coder/scripts/coder_format/coder_format.php \
  custom/annotate/annotate.module
```

Сценарий на месте модифицирует файл `annotate.module` и сохраняет оригинал под именем `annotate.module.coder.orig`. Чтобы просмотреть, какие изменения внес сценарий, воспользуйтесь утилитой `diff`:

```
$ diff custom/annotate/annotate.module custom/annotate/annotate.module.coder.orig
```

Поиск в коде с помощью `grep`

Утилита `grep` — одна из команд UNIX, осуществляющая в файлах поиск строк, которые соответствуют заданному регулярному выражению.

Пользователи Windows, которые желают ознакомиться приведенными ниже примерами, могут применять `grep`, установив уже скомпилированную версию (<http://unxutils.sourceforge.net>) либо установив среду Cygwin (<http://cygwin.com>). В противном случае вместо `grep` можно просто использовать встроенные функциональные возможности поиска Windows.

`grep` — удобный инструмент для поиска реализаций хуков в ядре Drupal, обнаружения мест, где создаются сообщения об ошибках, и т.д. Рассмотрим несколько примеров применения `grep` из корневого каталога Drupal:

```
$ grep -rl 'hook_init' .
./authorize.php
./includes/common.inc
./modules/simpletest/tests/system_test.module
./modules/simpletest/tests/theme_test.module
./modules/simpletest/tests/theme.test
./modules/simpletest/tests/actions_loop_test.module
./modules/locale/locale.module
./modules/dblog/dblog.module
./modules/update/update.module
./modules/system/system.api.php
./modules/system/system.module
./modules/overlay/overlay.install
./modules/overlay/overlay.module
./update.php
./themes/engines/phptemplate/phptemplate.engine
```

В данном случае мы рекурсивно (`-r`) ищем экземпляры `hook_init` среди файлов Drupal, начиная с текущего каталога (`.`), и выводим имена соответствующих файлов (`-l`). Теперь взгляните на следующий пример:

```
$ grep -rn 'hook_init' .
./authorize.php:31: * avoid various unwanted operations, such as hook_init() and
./includes/common.inc:2697: * drupal_add_css() in a hook_init() implementation.
./includes/common.inc:2750: * theme .info files. Modules that add
stylesheets within
hook_init()
./includes/common.inc:3770: * drupal_add_css() in a hook_init() implementation.
./includes/common.inc:3810: * hook_init() implementations, or from other
code that ensures that the
./includes/common.inc:4829: // Initialize $_GET['q'] prior to invoking
hook_init().
./includes/common.inc:4835: // Prior to invoking hook_init(), initialize
the theme (potentially a custom
./includes/common.inc:4837: // - Modules with hook_init() implementations
that call theme() or
./modules/simpletest/tests/system_test.module:184: * Implements hook_init().
...
```

Здесь в файлах Drupal выполняется рекурсивный (-r) поиск строк, содержащих строку `hook_init`, и вывод найденных строки вместе с их номерами (-n). Затем поиск можно уточнить, передав его результаты по конвейеру в другой поиск. В следующем примере мы ищем случаи вхождения слова `poll` в результатах поиска предыдущего примера:

```
$grep -rn 'hook_init' . | grep 'dblog'
./modules/dblog/dblog.module:88: * Implements hook_init().
```

Другой способ уточнения поиска — применение `grep` с флагом `-v`, который означает “инвертировать соответствие”, т.е. выбрать строки, не соответствующие заданному выражению. Давайте найдем все вхождения слова `lock`, которые не являются частью слова `block` или `Block`:

```
$ egrep -rn 'lock' . | egrep -v '[B|b]lock'
./includes/common.inc:2548: // See if the semaphore is still locked.
./includes/database.mysql.inc:327:function db_lock_table($table) {
./includes/database.mysql.inc:332: * Unlock all locked tables.
```

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Выполнять кодирование в соответствии с соглашениями Drupal о кодировании.
- Документировать код так, чтобы комментарии могли многократно использоваться модулем API.
- Эффективно выполнять поиск в кодовой базе Drupal, используя утилиту `grep`.
- Узнавать высококлассных разработчиков Drupal по их приемам кодирования.

ГЛАВА 23

Оптимизация Drupal

Архитектура ядра Drupal является экономичной, и при ее построении, прежде всего, преследовалась цель гибкости. Однако гибкость дается не бесплатно. По мере роста количества модулей увеличивается сложность обслуживания запроса. Это означает увеличение нагрузки на сервер, и для сохранения легендарного быстродействия Drupal при росте популярности сайта приходится реализовать определенные стратегии. Правильное конфигурирование Drupal позволяет легко выдержать резкий рост числа посетителей. В этой главе речь пойдет о производительности и масштабируемости. *Производительность* характеризует то, насколько быстро сайт отвечает на запрос. *Масштабируемость* определяется тем, сколько одновременно поступающих запросов система может обработать, и обычно измеряется в запросах в секунду.

Эта глава разбита на два больших раздела, посвященные инструментам и технологиям, которые помогут повысить производительность сайта, после чего рассматриваются методы повышения производительности медленного сайта. Мы начнем с рассмотрения действий, которые следует предпринять еще до того, как производительность сайта снизится.

На заметку! Ценный вклад в написание этой главы привнесли Курт Грей (Kurt Gray) и команда из Aquia.

Кэширование — ключ к высокой производительности Drupal

Три секрета получения оптимальной производительности Drupal могут быть сформулированы так: кэширование, кэширование и еще раз кэширование. Каждый уровень стека сервера Drupal предоставляет свои опции кэширования, и следует научиться получать максимальную выгоду от всех из них. Ниже приведен перечень ключевых областей, которые следует рассмотреть при поиске возможностей повышения производительности сайта.

- *Кэш кодов операций PHP.* Кэширование кодов операций чрезвычайно важно и его трудно переоценить. Не существует никакой веской причины отказа от кэширования кодов операций, если только вы по каким-либо соображениям не намерены мириться с высокими нагрузками сервера и замедлением процесса загрузки страниц. Для реализации кэширования кодов операций PHP можно использовать такие инструменты, как APC, XCache, eAccelerator и другие, любой из которых легко установить в среде PHP. Наилучший выбор для реализации кэширования кодов операций — APC (www.drupal.org/project/apc). Пример отчета, сгенерированного APC, приведен на рис. 23.1.

- *Инвертированный прокси-кэш.* Инвертированный прокси-кэш снимает с веб-серверов огромную часть нагрузки. Прокси-кэш — быстрый веб-сервер, который находится перед внутренними веб-серверами, кэшируя любой кэшируемый контент, проходящий через него (как в случае с кэшем со сквозной записью), в результате чего последующие веб-запросы обслуживаются непосредственно из прокси-кэша, а не из внутренних серверов. Немного позже мы рассмотрим Varnish — наиболее предпочтительное решение для инвертированного прокси-кэширования.
- *Кэши базы данных.* СУБД MySQL имеет собственные встроенные кэши, в частности, кэш запросов (`query_cache_size`) и кэш ввода-вывода файловой системы (`innodb_buffer_pool_size`), которые должны быть настолько велики, насколько позволяет доступный объем памяти сервера базы данных.
- *Кэши Drupal.* Среда Drupal поддерживает собственные кэши для страниц, блоков и представлений. Чтобы включить их, откройте страницу производительности Drupal в административном интерфейсе. В этой главе мы рассмотрим также Pressflow — оптимизированную версию Drupal, которая предоставляет усовершенствованные собственные механизмы внутреннего кэширования Drupal.



Рис. 23.1. Средство Alternative PHP Cache (APC) имеет интерфейс, позволяющий отображать распределение памяти и файлы, находящиеся в данный момент в кэше

Часто система сталкивается с падением производительности, когда данные должны быть перемещены на более медленное устройство, например, жесткий диск, или получены с него. Нельзя ли полностью обойти эту операцию для данных, которые сохранять не обязательно (вроде данных сеанса)? Для этого служит система memcached, которая читает и записывает данные в память. Memcached более сложна в настройке, чем другие решения, предложенные в этой главе, однако о ней стоит подумать, когда возникает потребность в улучшении показателей масштабируемости.

Drupal имеет встроенный кэш базы данных для кэширования страниц, меню и других данных Drupal, а СУБД MySQL способна кэшировать общие запросы, но как быть, если база данных подвергается чрезмерной нагрузке? Можно приобрести другой сервер базы данных либо просто разгрузить базу данных, сохраняя некоторые данные непосредственно в памяти, а не в базе данных. Библиотека memcached (www.danga.com/memcached/) и PHP-расширение PECL Memcache (<http://pecl.php.net/package/memcache>) — именно те средства, которые позволяют решить эти задачи.

Система memcached сохраняет произвольные данные в оперативной памяти и обслуживает данные с максимально возможной скоростью. Такой способ доставки данных работает быстрее, чем что-либо, зависящее от доступа к жесткому диску. Memcached хранит объекты и ссылается на них посредством уникальных ключей этих объектов. Определение того, какие объекты нужно помещать в memcached, зависит от программиста. Memcached ничего не известно о типе или природе того, что в нее помещается; с ее точки зрения все это — набор битов с ключами для извлечения.

Преимуществом системы memcached является ее простота. При написании кода для Drupal в расчете на использование memcached разработчики могут кэшировать все, что, по их мнению, является главными причинами возникновения узких мест. Это могут быть результаты часто выполняющихся запросов к базе данных, такие как поиски путей, или даже более сложные конструкции, вроде полностью построенных нод и словарей таксономии; те и другие требуют выполнения множества запросов к базе данных и значительных затрат на обработку PHP.

Модуль memcache для Drupal и специальный API-интерфейс для работы с интерфейсом PECL Memcache доступны по адресу <http://drupal.org/project/memcache>.

Оптимизация PHP

В серверах Apache существуют два способа выполнения кода PHP: Fastcgi (`mod_fcgid`, `mod_fastcgi` и PHP-FPM) либо `mod_php`. Основное различие между ними в том, что `mod_php` будет выполнять код PHP непосредственно в Apache, в то время как варианты Fastcgi будут передавать каждый PHP-запрос внешнему процессу `php-cgi`, который выполняет код PHP вне Apache, а затем возвращает свой вывод Apache по конвейеру.

В веб-сервере Nginx (он более подробно рассматривается далее в этой главе) выбор проще, поскольку он ограничен использованием только модуля `NginxHttpFcgiModule` (Fastcgi) и не имеет встроенного модуля интерпретатора PHP, такого как `mod_php`.

И `mod_php`, и варианты Fastcgi выполняют более или менее одно и то же — в конце концов, фактически они используют один и тот же базовый интерпретатор PHP, действующий под управлением одного и того же PHP-кода. Единственное важное различие между ними связано с тем, куда перенаправляются их входные и выходные данные. Поэтому не удивительно, что сравнительные тесты равных по размерам пулов процессов `mod_php` и Fastcgi дают примерно одинаковые результаты по нагрузкам сервера и производительности предоставления данных сайтом Drupal. Пул процессов Apache с модулем `mod_php` и 25 дочерними процессами и пул процессов Apache с модулем Fastcgi и 25 процессами PHP будут иметь одинаковые общие характеристики отпечатков памяти и производительности. Однако варианты Fastcgi предоставляют возможность опре-

деления размеров пула процессов PHP независимо от размера пула процессов Apache, в то время как при использовании `mod_php` объем пула интерпретаторов PHP равен количеству процессов Apache. Поэтому кое-кто может отдавать предпочтение подходу с применением `Fastcgi` по сравнению с использованием `mod_php`, т.к. `Fastcgi` “экономит память”. Это рассуждение могло бы быть справедливым, если не учитывать соображения по поводу размера кэша кодов операций APC (которые также рассмотрены в этой главе) и ограничить общее количество процессов `Fastcgi` числом, значительно меньшим числа дочерних процессов Apache. Однако жесткое ограничение размера пула процессов PHP может самым пагубным образом сказаться на пропускной способности PHP: это было бы равносильно закрытию трех полос загруженной четырехполосной скоростной магистрали под смехотворным предлогом “экономии места” и, тем самым, созданию пробок.

Существует еще одно важное соображение по поводу использования памяти: кэш кодов операций APC PHP разделяется процессами `mod_php` (все процессы `mod_php` ссылаются на один и тот же блок кэша APC), но в случае применения `mod_fcgid` кэш APC не разделяется процессами `php-cgi`. Учитывая, что типичный размер кэша кодов операций APC, выделенный для сервера Drupal, может составлять 50 Мбайт и более, при использовании кэша операций APC (что должно иметь место на любом серьезном сервере Drupal) весь пул процессов Apache и `php-cgi` в любом случае будет требовать значительно больше памяти, чем пул процессов Apache и `mod_php` такого же размера.

Какой же подход обеспечивает более высокую производительность? В действительности ни `mod_php`, ни `Fastcgi` не дают существенного выигрыша в производительности в сравнении друг с другом при условии выделения им одинакового объема ресурсов. Однако применение `Fastcgi` может быть рациональным, если по каким-либо причинам, например, в ситуации аренды веб-серверов несколькими пользователями, требуется изменять размеры пула процессов Apache независимо от пула процессов PHP, поскольку `Fastcgi` предоставляет возможность разделения процессов на уровне пользователей.

Установка файла кэша кодов операций PHP в `/dev/zero`

И APC, и XCache предоставляют возможность установки пути кэша кодов операций. В APC путь хранилища кэша — настройка `apc.mmap_file_mask` — определяет используемый механизм разделяемой памяти. Разделяемая память System V IPC — приемлемый выбор, но в большинстве систем Linux ее объем ограничен только 32 Мбайт; это значение может быть увеличено, но выделяемого по умолчанию объема недостаточно для кэша кодов операций типичных сайтов Drupal. Разделяемая память `mmap` POSIX допускает совместное использование блоков памяти любого размера. Однако ее производительность значительно снижается при подкачке посредством дискового файла, поскольку часто выполняемые операции ввода-вывода разделяемой памяти будут преобразовываться в объемные и частые операции дискового ввода-вывода, что особенно заметно на медленных дисках.

Решение состоит в установке пути карты памяти в `/dev/zero`, что указывает `mmap` на нежелательность дополнения области памяти дисковым хранилищем. К счастью, APC применяет этот режим по умолчанию, если только маска `apc.mmap_file_mask` явно не установлена в любой другой путь, отличный от `/dev/zero`.

Настройки пула процессов PHP

Под пулом процессов PHP здесь понимается весь пул процессов выполнения PHP на веб-сервере, определяющий количество одновременных PHP-запросов, которые сервер может обслуживать, не помещая запросы в очередь. Управление пулом процессов

осуществляется либо Apache и `mod_php`, либо какой-то вариацией Fastcgi: `mod_fcgid`, `mod_fastcgi` или PHP-FPM (FastCGI Process Manager). При настройке пула процессов PHP следует руководствоваться перечисленными ниже соображениями.

- Запускайте столько интерпретаторов PHP, сколько позволит доступный объем памяти. Если вы используете `mod_php`, то размер пула PHP равен количеству дочерних процессов Apache, определяемому настройками конфигурации Apache `StartServers`, `MinSpareServers`, `MaxSpareServers` и `MaxClients`, которые можно установить одинаковыми для обеспечения постоянства размера пула. В случае применения одного из вариантов Fastcgi, такого как `mod_fcgid`, размеры пула PHP `MaxProcessCount`, `DefaultMaxClassProcessCount` и `DefaultMinClassProcessCount` должны быть установлены одинаковыми для сохранения постоянства размера пула. Для веб-сервера с объемом памяти 8 Гбайт можно попробовать установить размер пула процессов PHP в 50, после чего подвергнуть сервер нагрузочному тесту, запросив множество различных страниц Drupal при числе одновременно обслуживаемых пользовательских клиентов 50 и времени обдумывания между запросами страниц, равном, по меньшей мере, 1 секунде для каждого клиента. Если сервер начнет испытывать нехватку памяти и/или выполнять подкачку с диска, необходимо уменьшить размер пула процессов PHP и повторить попытку. Во время подобного теста нагрузка сервера неизбежно возрастет, но в ходе данного настроечного теста это не имеет особого значения.
- Как можно дольше сохраняйте максимальное количество бездействующих интерпретаторов PHP. Желательно избежать “перемешивания” пула процессов PHP — т.е. постоянного удаления и воссоздания интерпретаторов PHP в ответ на изменение веб-трафика. Лучше создать максимальный пул интерпретаторов PHP, допуская объем памяти сервера, оставляя его постоянным даже при условии, что большую часть времени большинство этих процессов остается бездействующими. Для `mod_php` настройки `StartServers`, `MinSpareServers`, `MaxSpareServers` и `MaxClients` в Apache следует установить одинаковыми. В этом случае 50 — вполне подходящее начальное значение для веб-сервера Drupal с объемом памяти 8 Гбайт. Это создаст заранее подготовленный пул процессов Apache плюс `mod_php` постоянного размера. Еще одна ключевая настройка Apache для `mod_php` — `MaxRequestsPerChild`, которую в идеале следует устанавливать в 0, чтобы Apache заново не порождал дочерние процессы. Но если постепенно веб-сервер начинает испытывать утечку памяти, и есть веские причины полагать, что она связана с `mod_php`, значение `MaxRequestsPerChild` можно установить в 10000 или более, а затем постепенно снижать его, пока утечка памяти не будет устранена.
- При использовании `mod_fcgid` в случае возникновения ошибки сегментации при каждом 501-м PHP-запросе (известная программная ошибка `mod_fcgid`, которая, возможно, уже была исправлена на момент написания этой книги), потребуются установить значение `MaxRequestsPerProcess` в 500, что вынудит каждый интерпретатор `php-cgi` повторно порождать себя через каждые 500 запросов. В остальных случаях `MaxRequestsPerProcess` `mod_fcgid` следует устанавливать в 0, если только процессы `php-cgi` не приводят к утечке памяти.
- Кроме того, для `mod_fcgid` значения `IdleTimeout` и `IdleScanInterval` должны быть установлены равными нескольким часам или более во избежание накладных расходов, связанных с повторным порождением интерпретаторов PHP по требованию.

Настройка Apache

Существуют несколько параметров конфигурации, которые способствуют ускорению выполнения запросов сайтов Drupal, действующих на веб-сервере Apache. Следующие рекомендации помогут провести ряд наиболее существенных усовершенствований.

Модуль `mod_expires`

Этот модуль Apache позволяет Drupal отправлять HTTP-заголовки `Expires`, кэшируя все статические файлы в пользовательском браузере в течение двух недель либо до появления новых версий файлов. Это касается всех файлов изображений, CSS и JavaScript, а также других статических файлов. Результатом будет оптимизация полосы пропускания и меньший трафик, необходимый для взаимодействия с веб-сервером. Среда Drupal предварительно сконфигурирована на работу с модулем `mod_expires` и будет использовать его, если он доступен. Настройки для `mod_expires` находятся в файле `.htaccess` среды Drupal.

```
# mod_expires должен быть включен.
<IfModule mod_expires.c="">
    # Разрешить истечение срока.
    ExpiresActive On

    # Кэшировать все файлы сроком на 2 недели после доступа (A) .
    ExpiresDefault A1209600

    # Не кэшировать динамически генерируемые страницы.
    ExpiresByType text/html A1
</IfModule>
```

Мы не можем позволить `mod_expires` кэшировать HTML-контент, потому что HTML-контент, генерируемый Drupal, не всегда имеет статическую природу. Именно по этой причине Drupal поддерживает собственную внутреннюю систему кэширования HTML-вывода (т.е. кэширование страниц).

Перенос директив из `.htaccess` в `httpd.conf`

Среда Drupal поставляется с двумя файлами `.htaccess`: один находится в корневом каталоге Drupal, а другой автоматически генерируется после создания каталога для хранения загруженных на сервер файлов и указания на странице Configuration⇒File system (Конфигурация сайта⇒Файловая система) его местоположения. Поиск, чтение и синтаксический разбор любых файлов `.htaccess` происходит при каждом запросе. С другой стороны, файл `httpd.conf` считывается только при запуске Apache. Директивы Apache могут размещаться в любом из этих файлов. Если вы управляете собственным сервером, можно перенести содержимое файлов `.htaccess` в главный файл конфигурации Apache (`httpd.conf`) и запретить поиск файлов `.htaccess` в пределах корневого каталога веб-сервера, указав для `AllowOverride` значение `None`:

```
<Directory />
    AllowOverride None
    ...
</Directory>
```

Это предотвращает просмотр дерева каталогов сервером Apache при каждом запросе в поисках файла `.htaccess` для его выполнения. В результате Apache выполняет меньший объем работы по обслуживанию каждого запроса, что позволяет ему обслуживать большее количество запросов.

Сравнение предварительного ветвления MPM с рабочей конфигурацией Apache MPM

Выбор между предварительным ветвлением Apache (prefork) и рабочей конфигурацией (worker) ведет к использованию нескольких дочерних процессов Apache либо меньшего количества дочерних процессов, каждый из которых использует несколько потоков. В общем случае для Drupal лучше применять предварительное ветвление Apache. Причины объясняются ниже.

PHP не обеспечивает безопасную обработку потоков, поэтому при использовании `mod_php` единственный реальный вариант выбора — предварительное ветвление Apache. При использовании варианта Fastcgi (такого как `mod_fastcgi` или `mod_fcgid`), можно применять рабочую конфигурацию MPM Apache, поскольку PHP-запросы будут обрабатываться внешне из Apache.

Однако использование рабочей конфигурации MPM Apache вместо предварительного ветвления MPM Apache обеспечит не столь уж большой выигрыш, как можно было бы думать, полагая, что потоки автоматически делают многопоточные приложения более быстродействующими и масштабируемыми, чем их заранее разветвленные многопроцессорные эквиваленты, даже в многоядерных системах. Это объясняется несколькими причинами.

Во-первых, следует понимать, что в действительности представляют собой потоки в операционной системе Linux: по большей части — это те же дочерние процессы. Их различает лишь то, что поток обладает прямым общим доступом к содержимому памяти родительского процесса, в то время как разветвленный дочерний процесс получает ссылку для копирования при записи на содержимое своего родительского процесса. Это различие обеспечивает определенное преимущество потокам, которое легко утрачивается вследствие зачастую сложной логики синхронизации общего доступа к памяти между потоками.

Во-вторых, интуитивное представление о том, что потоки используют значительно меньший объем памяти, чем отдельные дочерние процессы, не соответствует действительности. Применение общих системных инструментов, таких как `top` и `ps`, создает впечатление, что каждый дочерний процесс Apache использует почти столько же памяти, как и его родительский процесс. Фактически большая часть отиска памяти каждого дочернего процесса Apache представляет собой те же области памяти, которые используются родительским процессом Apache, учитываемые несколько раз. Это связано с тем, что большую часть отиска памяти дочерних процессов составляет содержимое общих библиотек, которые большинство операционных систем, будучи достаточно интеллектуальными, загружают в память только единожды, и каждый дополнительный процесс, применяющий те же самые библиотеки, ссылается на первую общую копию в памяти. Еще одно соображение по поводу использования памяти заключается в том, что дочерние процессы будут разделять большую часть содержимого памяти своих родительских процессов, если только не будут модифицировать это содержимое (копировать при записи).

В-третьих, можно остановить вышедшие из-под контроля дочерние процессы Apache, но не вышедшие из-под контроля потоки Apache, не перезапуская весь сервер Apache. С точки зрения администратора сервера, проблемы проще диагностировать и устранять в предварительно разветвленном пуле процессов Apache, чем в многопоточном пуле процессов.

Конечно, конкретные характеристики системы могут быть различными, поэтому все же имеет смысл провести сравнительные тесты различных конфигураций MPM Apache.

Балансировка размера пула Apache

При использовании предварительного ветвления Apache размер пула дочерних процессов Apache желательно определить так, чтобы предотвратить переполнение пула процессов. Иначе говоря, при запуске сервера Apache желательно немедленно создать предварительно разветвленный пул процессов Apache (содержащий столько процессов, сколько может поддерживать память веб-сервера), чтобы этот весь пул дочерних процессов существовал и ожидал запросы, даже если большую часть времени они бездействуют, вместо постоянного устранения перегрузки за счет уничтожения и повторного порождения дочерних процессов Apache в ответ на сиюминутные колебания трафика.

Ниже приведен пример настроек предварительного ветвления Apache для веб-сервера Drupal с модулем `mod_php`:

```
StartServers 40
MinSpareServers 40
MaxSpareServers 40
MaxClients 80
MaxRequestsPerChild 20000
```

Эти настройки указывают Apache немедленно запустить 40 дочерних процессов и оставлять это значение неизменным даже при низком объеме трафика, но повышать его до 80 дочерних процессов при значительном возрастании трафика. (Пределы 40 и 80 можно увеличить в соответствии с масштабом реального сервера.)

При взгляде на эти настройки может возникать вопрос, не является ли бесполезной тратой памяти существование множества бездействующих процессов Apache? Не забывайте, что наша главная цель — обеспечение быстрой доставки страницы, а существование большого объема свободной памяти не сулит никакой выгоды. Какая польза от заявлений вроде “мой сервер работает медленно, но взгляните, сколько в нем свободного ОЗУ”? Если у вас есть память, так используйте ее!

Снижение длительности тайм-аута Apache

Настройка `Timeout` (тайм-аут) в конфигурации Apache определяет длительность интервала времени, в течение которого веб-клиент может оставлять подключение открытым, не предпринимая никаких действий. По умолчанию значение `Timeout` в Apache равно 5 минутам (300 секундам), а это слишком много. Уменьшите это значение до 20 секунд или даже меньше.

Отключение неиспользуемых модулей Apache

Закомментируйте любые директивы загрузки модулей `LoadModule` в Apache, если уверены, что они не нужны. К кандидатам на отключение относятся модули `mod_cgi`, `mod_dav` и `mod_ldap`.

Использование Nginx вместо Apache

Наиболее предприимчивые администраторы LAMP заменяют Apache на Nginx. Как известно, Nginx — замечательный сервер общего назначения с огромными возможностями в плане масштабирования. Однако Nginx не поддерживает `mod_php` — вместо него приходится ограничиваться использованием `Fastcgi` (`php-cgi`) для обслуживания PHP-запросов, что само по себе является не чем-то плохим, а просто другим подходом. Кроме того, Nginx не воспринимает Apache-файлы `.htaccess`, поэтому любые специфичные для `.htaccess` директивы в кодовой базе Drupal, такие как `Boost cache`, приходится преобразовывать в эквивалентные директивы конфигурации Nginx.

Многие склонны считать Nginx более быстрым сервером. Но в любом стеке Drupal реальным узким местом является PHP или уровень базы данных, а не выбор того или иного веб-сервера. Тем не менее, достоинства Nginx делают его хорошим кандидатом на роль балансировщика нагрузки и сервера статического контента.

Использование Pressflow

Pressflow — это внедряемая замена стандартного ядра Drupal, которая включает множество усовершенствований производительности по сравнению с ядром Drupal. В остальном, несмотря на явные внешние отличия, Pressflow является целиком все той же средой Drupal. Многие функциональные средства Pressflow продолжают внедряться в ядро Drupal; однако специалисты из компании Four Kitchens продолжают пытаться выйти за границы возможного, занимаясь оптимизацией Drupal. На момент написания этой книги официального выпуска Pressflow для Drupal 7 не существовало. Актуальная информация по средствам и функциональности Pressflow доступна на веб-сайте www.pressflow.org.

Использование Varnish

Varnish становится предпочитаемым кэширующим прокси-сервером в сообществе Drupal. Он представляет собой быстрый и мощный инвертированный кэширующий прокси-сервер HTTP. Типовой сервер приложений Drupal способен доставлять сотни динамических страниц Drupal в минуту. Varnish обеспечивает возможность доставки тысяч кэшированных страниц Drupal в секунду! Более того, запросы, обслуживаемые из Varnish, не создают никакой нагрузки для внутренних серверов, поскольку доставляемые из кэша запросы никогда не достигают внутренних серверов.

В типичной конфигурации Varnish устанавливается для прослушивания порта 80 (стандартный прослушивающий порт веб-сервера), поэтому все запросы веб-контента вначале попадают в Varnish. Сервер Varnish решает, нужно ли обслуживать запрос непосредственно из собственного кэша или повторить запрос для внутренних веб-серверов. Политики кэширования и доставки определены в локальном файле конфигурации VCL (Varnish Configuration Language — язык конфигурации Varnish).

Язык VCL предоставляет администраторам Varnish возможность определения чрезвычайно специфичных политик кэширования, используя условные выражения подобных выражениям Javascript. Язык VCL предоставляет также возможности балансировки нагрузки запросов между множеством внутренних серверов, перезаписывания запросов, изменения их содержимого и блокирования запросов. Более того, VCL позволяет включать встроенный код на языке C для тех, кто желает управлять процессом доставки запросов на самых низких уровнях.

Обратите внимание, что Varnish не поддерживает протокол SSL (HTTPS-запросы) и не предоставляет отдельные конфигурации виртуальных хостов в разделяемой среде хостинга. Однако выражения VCL Varnish могут быть помещены внутрь условных конструкций в зависимости от целевого хоста запроса.

Следует также отметить, что Varnish представляет собой кэш со сквозной записью HTTP, а не обобщенное хранилище данных “ключ/значение” и поэтому не подходит для кэширования памяти и не предоставляет прямой API-интерфейс для хранения и загрузки произвольных данных из кэша.

Другими альтернативами прокси-кэша HTTP являются Squid, Apache в сочетании с `mod_cache` и модуль прокси-кэша HTTP Nginx. Однако эти варианты не предлагают того разнообразия возможностей, которые предоставляет язык VCL Varnish.

Следует отметить, что Varnish является многопоточным, поэтому его масштабируемость ограничена количеством потоков сервера Varnish, который данный сервер может управлять одновременно. Умеренно загруженный сервер Varnish может содержать несколько сотен действующих потоков, а в сильно загруженном сервере Varnish число потоков может превышать тысячу. Если Varnish не способен порождать дополнительные потоки, дополнительные запросы веб-сайта будут сталкиваться с ошибками типа “Connection reset” (Сброс подключения).

Чтобы позволить Varnish порождать больше потоков, отредактируйте сценарии запуска Varnish, откорректировав опции `-w` (опции рабочего пула потоков), которые передаются команде запуска Varnish. Второй параметр, переданный в опции `-w`, представляет максимальное количество потоков, которые может породить Varnish. Увеличьте его значение, по крайней мере, до 4000.

Кроме того, в системах Linux по умолчанию каждому потоку выделяется 8 Мбайт виртуальной памяти, что намного превышает тот объем, который будет требоваться любому потоку Varnish. Поэтому в сценарий запуска Varnish потребуется добавить команду `ulimit -s 512`, чтобы уменьшить объем стека, по умолчанию резервируемый для одного потока, до 512 Кбайт.

Нормализация входящих запросов для увеличения коэффициента попаданий в кэш Varnish

Ключ к достижению высоких коэффициентов попадания в кэш Varnish — нормализация входящих HTTP-запросов так, чтобы все анонимные запросы одного и того же URL приводили к тому же самому попаданию в кэш из Varnish.

Чтобы разобраться с синхронизацией кэша Varnish, вначале нужно понять, как Varnish сохраняет записи кэша для каждого URL. Сервер Varnish объединяет следующие атрибуты запроса в хеш-ключ, который он использует для хранения и поиска записей своего кэша:

- URL запроса
- входящий заголовок Host
- входящий заголовок Cookie
- входящий заголовок Accept-Encoding

Проблема в том, что заголовки Cookie и Accept-Encoding изменяются от браузера к браузеру. Например, весьма вероятно, что множество браузеров, попадающих на ваш веб-сайт, обладают различными cookie-наборами и, следовательно, различными заголовками Cookie. Для нивелирования этого различия между входящими заголовками Cookie желательно (и это лучше всего) удалить весь заголовок Cookie на этапе `vcl_recv` конфигурирования Varnish, как показано в следующем примере:

```
sub vcl_recv {
    # Удалить входящий заголовок Cookie из анонимных запросов.
    if (req.http.Cookie !~ "(^|;\s*)SESS") {
        unset req.http.Cookie;
    }
    # ... другие правила vcl_recv ...
    # Не предоставлять кэшированный контент для зарегистрированных пользователей.
    if (req.http.cookie ~ "SESS") {
        return (pass);
    }
    # Попытка обслуживания из кэша
    return (lookup);
}
```

Приведенный фрагмент VCL-кода проверяет, поступает ли запрос от зарегистрированного пользователя (заголовок `Cookie` которого начинается с `SESS`) и, если это не так, нормализует заголовок `Cookie`, полностью его удаляя. Если некоторые cookie-наборы анонимных запросов требуется продублировать внутренним серверам, можно настроить отсечение заголовков `Cookie` либо добавить несколько строк, уточняющих, какие cookie-наборы должны обходить этап поиска в кэше Varnish.

Еще один заголовок входящего запроса, нуждающийся в нормализации — `Accept-Encoding`, поскольку он несколько отличается в разных типах браузеров. Наиболее часто этот заголовок применяется для уведомления браузером веб-сервера о том, что браузер может принимать сжатый контент. Типичный фрагмент VCL-кода нормализации заголовков `Accept-Encoding` выглядит следующим образом:

```
# Нормализовать заголовки Accept-Encoding для обеспечения лучшей
когерентности кэша.
if (req.http.Accept-Encoding) {
    # Нет смысла сжимать медиа-данные, которые уже сжаты.
    if (req.url ~ "\.(jpg|png|gif|gz|tgz|bz2|tbz|mp3|ogg)$") {
        remove req.http.Accept-Encoding;
    }
    # Обход ошибки обработки JS в MSIE 6.
    } elseif (req.http.User-Agent ~ "MSIE 6") {
        unset req.http.Accept-Encoding;
    } elseif (req.http.Accept-Encoding ~ "gzip") {
        set req.http.Accept-Encoding = "gzip";
    } elseif (req.http.Accept-Encoding ~ "deflate") {
        set req.http.Accept-Encoding = "deflate";
    } else {
        # Неизвестный алгоритм.
        remove req.http.Accept-Encoding;
    }
}
```

Varnish: обнаружение внешних cookie-наборов

Следующая командная строка в сервере Varnish полезна для отслеживания актуальных поступающих заголовков `Cookie`, которые дублируются из Varnish внутренним серверам:

```
varnishlog | grep TxHeader | grep Cookie
```

Она полезна для настройки фильтрации заголовков `Cookie` в Varnish.

Модуль Boost

Популярный модуль Boost для Drupal по существу строит статический файловый кэш для динамически генерируемого контента Drupal. Когда модуль Boost установлен в Drupal, при каждой генерации Drupal динамической страницы модуль Boost будет сохранять статическую копию этого контента, чтобы следующий анонимный запрос этой же страницы мог быть обслужен из кэша Boost. Периодически фоновый процесс `cron` удаляет из кэша Boost устаревшие страницы, которые затем восстанавливаются при следующем запросе. Этот подход снижает общую перегрузку PHP и MySQL, но все же требует, чтобы Apache (или Nginx, IIS, lighttpd) выполнял обработку нескольких дополнительных правил перезаписи при каждом запросе страницы.

Ключ к достижению высокой производительности модуля Boost — размещение каталога кэша в быстрой локальной файловой системе. Некоторые администраторы Drupal могут помещать файлы кэша Boost в сетевую разделяемую файловую систему, чтобы

множество веб-серверов могли совместно использовать одни и те же файлы кэша Boost. Однако на загруженном веб-сайте обслуживание кэша Boost может порождать множество операций ввода-вывода файловой системы, что ведет к значительному снижению быстродействия сетевой разделяемой файловой системы — в этом случае не остается ничего другого, кроме как размещение кэша Boost в качестве локального каталога на каждом веб-сервере.

Если каждый веб-сервер обладает достаточно большим объемом памяти, но медленными дисками, файлы кэша Boost можно записывать в локальную файловую систему `ramfs` — это средство Linux позволяет создавать эфемерный том хранилища, существующий исключительно в ОЗУ.

Сравнение Boost и Varnish

Хотя Boost и Varnish предлагают разные виды решений кэширования, администраторы Drupal часто непосредственно сравнивают их друг с другом. В целом, Boost проще в установке и администрировании, чем Varnish. Однако Varnish предоставляет общее решение по улучшению производительности, поскольку может использоваться для прокси-кэширования и других видов контента, таких как статические изображения и таблицы стилей, а не только страниц Drupal. Кроме того, Varnish также обеспечивает возможность балансировки нагрузки и перезаписи запросов еще до их попадания в веб-сервер, в то время как запросы Boost в любом случае попадают в него.

Тем не менее, Boost и Varnish можно использовать совместно. Для этого достаточно настроить заголовки срока действия своего кэша HTTP и очистку кэша Boost, чтобы Varnish и Boost своевременно очищали свои кэши.

Настройка системы Linux для серверов с высоким уровнем трафика

Тема настройки Linux для обеспечения больших объемов веб-трафика заслуживает отдельной книги. Однако существует ряд простых изменений, таких как представленные в следующем сценарии `sysctl_set.sh`, которые помогут повысить пропускную способность сайтов.

```
#!/bin/sh
# Tweaks (see http://varnish-cache.org/wiki/Performance)
echo "
net.ipv4.ip_local_port_range = 1024 65536
net.ipv4.tcp_rmem=4096 87380 16777216

net.ipv4.tcp_wmem=4096 65536 16777216
net.ipv4.tcp_fin_timeout = 3
net.ipv4.tcp_no_metrics_save=1
net.ipv4.tcp_syncookies = 0
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_synack_retries = 2

net.ipv4.tcp_syn_retries = 2
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.core.netdev_max_backlog = 30000
net.core.somaxconn = 262144
" > sysctl_tweaks.conf
sysctl -p sysctl_tweaks.conf
```

Назначение приведенных переменных описано ниже.

- *ip_local_port_range*. Устанавливает максимальное количество сетевых портов, доступных для установления сетевых подключений.
- *tcp_rmem* and *tcp_wmem*, *rmem_max* и *wmem_max*. Увеличивают размер буферов сетевого ввода-вывода.
- *tcp_fin_timeout*. Уменьшает время закрытия продолжительных сетевых подключений.
- *tcp_max_orphans*. Увеличивает количество поддерживаемых системой сокетов, которые еще не подсоединены к чему-либо.
- *tcp_max_syn_backlog*. Увеличивает число квитирований SYN, сохраняемых в памяти (требуется *tcp_syncookies=1*).
- *tcp_synack_retries*. Уменьшает количество попыток установления TCP-подключения.
- *netdev_max_backlog*. Увеличивает число входящих пакетов, которые могут быть помещены в очередь для обработки более высокого уровня.
- *somaxconn*. Устанавливает размер прослушиваемой очереди для приема новых TCP-подключений.

Использование быстрых файловых систем

Медленные файловые системы — неразрешимая проблема стеков LAMP. Каждый уровень LAMP очень часто обращается к файловой системе. Хранение базы данных в медленной файловой системе неизбежно приведет к снижению производительности. Примеры быстрых файловых систем включают:

- *ramfs* или *tmpfs* (используют память вместо дискового пространства);
- *ext2* на локальном диске;
- *ext3* на локальном диске;
- *xfs* на локальном диске;
- аппаратный массив RAID;
- система SAN или NAS, использующая специализированное оборудование.

А вот примеры медленных файловых систем (в сравнении с перечисленными выше):

- виртуальные диски (внутри среды любого виртуализированного сервера);
- NFS и другие типы программно управляемых сетевых файлов совместного использования;
- программные массивы RAID (в зависимости от выбранного уровня RAID);
- S3FS (монтирует хранилище Amazon S3 в качестве локального диска);
- LVM (замедление работы тем больше, чем больше сохраняется снимков системы).

В значительной мере архитектура стека LAMP зависит от выбора томов для хранения веб-контента и таблиц базы данных, исходя из размера, быстродействия и надежности файловой системы. Для достижения самой высокой производительности необходимо использовать наиболее быструю из доступных файловых систем и обеспечить безотказность и целостность за счет избыточности (т.е. нескольких серверов и дубликатов базы данных).

Сравнение выделенных и виртуальных серверов

Как правило, выделенные физические серверы работают быстрее виртуальных серверов, когда дело доходит до сетевого ввода-вывода, дискового ввода-вывода и ввода-вывода в памяти, даже в ситуациях, при которых виртуальному серверу теоретически предоставлено больше ресурсов (времени ЦП, дискового пространства и памяти), чем выделенному серверу аналогичной конфигурации. При таком сравнении важно учитывать, что в среде виртуализированного сервера загрузка ЦП, дисковый ввод-вывод, ввод-вывод в памяти и сетевой ввод-вывод неизбежно связаны с уровнями маршрутизации ввода-вывода между операционной системой и фактическим оборудованием сервера. И, следовательно, все операции ввода-вывода подвергаются их постановке в очередь гипервизором хоста, а также зависят от потребностей соседних компьютеров этого же физического хоста.

Например, виртуальный сервер, содержащий сервер базы данных может располагать вдвое более быстродействующим ЦП, чем выделенный физический сервер, однако виртуальный сервер может добавлять 1 мс сетевой задержки (весьма реальный пример виртуализированной среды Xen) при обмене данными даже между соседними компьютерами. Сетевая задержка, равная 1 мс, кажется не слишком существенной, но если учесть, что зарегистрированный запрос страницы Drupal может быть сопряжен с сотнями упорядоченных запросов MySQL, то в результате общая сетевая задержка загрузки страницы может превысить даже целую секунду. Дополнительная односекундная задержка на обслуживание запроса страницы, на первый взгляд, также представляется допустимой. Однако следует учитывать также частоту поступающих запросов страницы и то, не приведет ли эта односекундная задержка к тому, что процессы PHP образуют значительный трафик, тем самым, повышая нагрузку сервера. Добавление дополнительных и более мощных виртуальных серверов в стек также не устраняют этот фактор задержки ввода-вывода. Это же относится и к дисковому вводу-выводу: виртуальные диски всегда уступают в скорости локальным физическим дискам, независимо от выделенного виртуальному серверу времени ЦП и объема памяти.

Однако виртуальные серверы обладают преимуществом “эластичности”, т.е. их проще быстро масштабировать по горизонтали (добавляя дополнительные серверы). Кроме того, когда выделенное оборудование отказывает, его приходится останавливать и устранять неисправность, если только вы не располагаете множеством запасных серверов; к тому же, как всем нам хорошо известно, действительно оборудование с возможностью “горячей замены” — дорогое удовольствие, о котором системным администраторам можно только мечтать.

Предотвращение обращения к внешним веб-службам

Настоящим проклятием веб-сервера, с которыми приходится сталкиваться достаточно часто, являются нестандартные модули Drupal, обращающиеся к внешней веб-службе, которая оказывается медленной или просто не отвечает. Подобные элементы могут быстро сделать веб-сервер не реагирующим на запросы страниц, поскольку все процессы PHP устанут дожидаться внешней службы, которая не отвечает. Основная причина этой проблемы в том, что по умолчанию значение `default_socket_timeout` установлено в 60 секунд, в результате чего каждый процесс PHP будет блокировать соединение на целую минуту, дожидаясь пакета, который не поступает.

Первое, приходящее на ум решение — не делать этого, т.е. не выполнять частых вызовов внешней веб-службы, которая не поддается контролю, а вместо этого применять

какую-то иную стратегию, например, фоновый процесс, который периодически извлекает внешний контент и локально его кэширует. Но если обращение к внешней веб-службе все же необходимо, по крайней мере, используйте PHP-функцию `stream_set_timeout()` или уменьшите значение `default_socket_timeout` в файле `php.ini`, чтобы не отвечающие соединения разрывались через 3 секунды.

Уменьшение интервалов ожидания сервера

Каждый уровень стека сервера LAMP имеет ряд настроек интервалов тайм-аута. Уменьшение значений этих настроек позволяет предотвратить накопление процессов медленной или не отвечающей службы в веб-сервере. Значения всех интервалов тайм-аута рекомендуется уменьшить до минимально приемлемых.

Например, модуль `mod_fcgid` в Apache имеет настройку `BusyTimeout`, которая по умолчанию ожидает 5 минут, прежде чем прервать длительно выполняющийся процесс PHP. Ее значение можно уменьшить до 30 секунд, справедливо полагая, что любая страница, загрузка которой занимает более 30 секунд, должна быть просто признана недоступной, вместо того, чтобы загружать веб-сервер в течение еще 4 минут.

Другие важные интервалы тайм-аута, об уменьшении которых можно подумать — настройка `Timeout` в Apache, `max_execution_time` в PHP, `default_socket_timeout` в PHP, `proxy_read_timeout` в Nginx, а также ряд настроек TCP ядра Linux.

СУБД MySQL не предоставляет возможности настройки максимального времени выполнения запроса, но внешний процесс можно обязать вести список процессов `SHOW PROCESSLIST` (отобразить список процессов) и прерывать любые длительно выполняющиеся элементарные запросы.

Выделяющийся из общего ряда процесс PHP в Drupal, которому можно разрешить выполняться дольше 5 минут — `cron.php`, который инициирует все обращения к `hook_cron()` из Drupal. Рекомендуется делегировать `hook_cron()` только быстро выполняемые простые задачи, а задачи посложнее — сценариям оболочки `crontab`.

Оптимизация базы данных

Среда Drupal интенсивно взаимодействует с базой данных, особенно для аутентифицированных пользователей и специализированных модулей. СУБД очень часто становятся узким местом. Ниже описано несколько стратегий оптимизации использования базы данных в Drupal.

Включение кэша запросов в MySQL

СУБД MySQL применяется вместе с Drupal чаще других. В MySQL имеется возможность кэшировать часто встречающиеся запросы в оперативной памяти, так что при следующем поступлении конкретного запроса MySQL возвратит его непосредственно из кэша. Однако в большинстве инсталляций MySQL эта возможность *по умолчанию отключена*. Чтобы ее активировать, добавьте приведенные ниже строки в файл настроек MySQL; этот файл называется `my.cnf` и определяет переменные и поведение сервера MySQL (см. <http://dev.mysql.com/doc/refman/5.1/en/option-files.html>). В данном случае кэш запросов устанавливается равным 64 Мбайт:

```
# The MySQL server
[mysqld]
query_cache_size=64M
```

Текущий объем кэша запросов можно просмотреть с помощью MySQL-команды `SHOW VARIABLES:`

```
mysql>SHOW VARIABLES LIKE 'query_cache%';
...
| query_cache_size          | 67108864
| query_cache_type          | ON
...
```

Обычно нужно поэкспериментировать с размером кэша запросов. Слишком маленький кэш приведет к тому, что кэшируемые запросы очень часто будут оказываться недействительными. Чересчур большой кэш повлечет за собой относительно длительный поиск в кэше; вдобавок память, попусту расходуемая для кэша, может более эффективно использоваться для других нужд, например, для увеличения количества процессов веб-сервера, включения системы memcached или кэширования файла операционной системы.

Совет. В Drupal перейдите на страницу Reports⇒Status report (Отчеты⇒Отчет о состоянии) и щелкните на номере версии MySQL, чтобы получить краткий обзор значений некоторых наиболее важных переменных MySQL. На этой странице можно также проверить, включено ли кэширование запросов.

Производительность InnoDB MySQL в среде Windows

Механизм хранилища InnoDB MySQL, который по умолчанию применяется в Drupal при использовании MySQL, обладает особенно низкой производительностью записи в среде Windows. Эта низкая производительность проявится в Drupal, когда при попытке загрузки страницы Admin Modules (Модули администратора) выяснится, что до окончания загрузки вполне можно успеть соорудить себе бутерброд. Эту проблему можно решить двумя способами: либо преобразовать все таблицы в MyISAM (это решение подходит для серверов с низким уровнем трафика), либо в конфигурации MySQL установить настройку `innodb_flush_log_at_trx_commit=2`, указывающую InnoDB, что не следует дожидаться завершения операций записи на диск.

Производительность Drupal

Существуют две часто недооцениваемые области повышения производительности Drupal, которые легко реализовать.

Предотвращение ошибок 404

Одна из наиболее недооцениваемых причин снижения производительности типичного сайта Drupal — внешне безобидные ошибки 404 (File not found — Файл не найден). Это связано с тем, что часто Drupal конфигурируется для предоставления полного динамического ответа на ошибку 404, даже если запрос касается крошечного файла изображения в забытой таблице стилей или давно удаленной пиктограммы `favicon.ico`.

Решение заключается с устраниении каждой из ошибок 404, отраженных в административных журналах Drupal, и изменении директивы `ErrorDocument` в файле `.htaccess`, чтобы она выглядела примерно так:

```
<FilesMatch "\.(png|gif|jpe?g|s?html?|css|js|cgi|ico|swf|flv|dll)$">
  ErrorDocument 404 default
</FilesMatch>
```

Отключение неиспользуемых модулей

В процессе каждой начальной загрузки Drupal будет просматривать подкаталоги всех своих модулей и загружать PHP-код любого обнаруженного модуля. Поэтому любые не используемые активно модули лучше полностью удалить из дерева исходных кодов Drupal.

Специфичные для Drupal методы оптимизации

Хотя большинство способов оптимизации Drupal реализуется на других уровнях стека программного обеспечения, существует несколько механизмов непосредственно в Drupal, которые могут помочь значительно увеличить производительность.

Кэширование страниц

Иногда простые вещи почему-то забываются, поэтому о них нужно напоминать. Drupal имеет встроенный механизм сокращения нагрузки на базу данных, основанный на хранении и пересылке сжатых кэшированных страниц, запрошенных анонимными пользователями. Включение кэширования по существу уменьшает количество запросов базы данных, необходимых для загрузки страницы, до одного, вместо множества запросов, которые выполнялись бы в противном случае. По умолчанию кэширование в Drupal отключено и может быть сконфигурировано на странице Configuration⇒Performance (Конфигурация⇒Производительность). Более подробно кэширование страниц описано в главе 16.

Оптимизация полосы пропускания

На странице Configuration⇒Performance доступна и другая возможность оптимизации производительности — уменьшение количества запросов, отправляемых серверу. При включенной функции Aggregate and compress CSS files into one (Объединять и сжимать файлы CSS в один) Drupal будет сжимать файлы CSS, созданные модулями, и упаковывать их в единственный файл внутри каталога css системного пути. Функция Aggregate JavaScript files into one file (Объединять файлы JavaScript в один файл) позволяет объединять множество файлов JavaScript в один и помещать этот файл внутрь каталога js системного пути. Это уменьшает количество HTTP-запросов на страницу и общий объем загруженной страницы.

Усечение таблицы сеансов

Drupal хранит пользовательские сеансы в своей базе данных, а не в файлах (см. главу 17). Это упрощает установку Drupal на множество машин, но также увеличивает накладные расходы базы данных, связанные с необходимостью управлять информацией о сеансе каждого пользователя. Если десятки тысяч пользователей посещают сайт ежедневно, очень быстро эта таблица может стать просто огромной.

PHP позволяет управлять тем, насколько часто будут удаляться записи старых сеансов. Drupal предоставляет возможность этой конфигурации в своем файле settings.php:

```
ini_set('session.gc_maxlifetime', 200000); // 55 часов (в секундах)
```

По умолчанию система очищает старые записи через интервалы, несколько превышающие два дня. Это означает, что если пользователь не войдет в систему в течение двух дней, его сеанс будет удален. Если размер таблицы sessions неуклонно увеличивается, частоту очистки PHP-сеансов нужно повысить.


```
ini_set('session.gc_maxlifetime', 86400); // 24 часа (в секундах)
ini_set('session.cache_expire', 1440); // 24 часа (в минутах)
```

При настройке значения `session.gc_maxlifetime` целесообразно использовать это же значение и для настройки `session.cache_expire`, которая управляет сроком хранения кэшированных страниц сеанса. Обратите внимание, что значение `session.cache_expire` задается в минутах.

Управление трафиком аутентифицированных пользователей

Поскольку Drupal может предоставлять кэшированные страницы для анонимных пользователей, которым обычно не требуются интерактивные компоненты Drupal, можно уменьшить промежуток времени, в течение которого эти пользователи могут оставаться зарегистрированными на сайте, или даже принудительно отменять их регистрацию сразу же после закрытия ими окна браузера. Это осуществляется посредством настройки времени жизни cookie-наборов в файле `settings.php`. В следующей строке это значение изменяется на 24 часа:

```
ini_set('session.cookie_lifetime', 86400); // 24 часа (в секундах)
```

А вот как пользователи принудительно выводятся из сайта, когда закрывают окно браузера:

```
ini_set('session.cookie_lifetime', 0); // Когда окно браузера закрывается.
```

Значение по умолчанию в `settings.php` (2 000 000 секунд) позволяет пользователю оставаться на сайте немногим дольше трех недель (если строки их сеансов не будут удалены из базы данных `sessions` в результате уборки мусора).

Протоколирование в базе данных

Среда Drupal поставляется с модулем протоколирования в базе данных (`dblog`), включенным по умолчанию. Записи журнала можно просмотреть на странице `Reports` ⇒ `Recent log entries` (Отчеты ⇒ Последние записи в журнале). Таблица `watchdog` в базе данных, содержащая журнальные записи, может очень быстро разрастаться, если ее регулярно не чистить. Когда выясняется, что большой размер таблицы `watchdog` замедляет работу сайта, можно сохранять ее небольшой, подкорректировав настройки на странице `Configuration` ⇒ `Logging and errors` (Конфигурация ⇒ Протоколирование и ошибки). Обратите внимание, что изменение этой настройки вступит в силу только при следующем запуске `cron`. Отсутствие регулярного запуска `cron` позволит таблице `watchdog` расти бесконечно, порождая существенные накладные расходы.

Протоколирование в журнале операционной системы

Модуль `syslog`, поставляемый в составе ядра Drupal, но отключенный по умолчанию, посредством PHP-функции `syslog()` записывает вызовы `watchdog()` в журнал операционной системы. Это подход исключает необходимость операций вставки в базу данных, требуемых модулем протоколирования Drupal.

Запуск cron

Несмотря на то что настройка `cron` является седьмым шагом в инструкции по установке Drupal, ее часто пропускают, и эта небрежность может поставить сайт в непростые условия. Если `cron` не запускается на сайте Drupal, база данных заполняется журнальными записями, устаревшими записями кэша и другими статистическими данными, которые бы в противном случае регулярно удалялись из системы. Поэтому настоятельно рекомендуется конфигурировать `cron` с самого начала в ходе нормаль-

ного процесса инсталляции. Для получения дополнительной информации по настройке `cron` просмотрите седьмой шаг в файле `INSTALL.txt`.

Совет. Если вы находитесь в критической ситуации, когда `cron` вообще не запускался на сайте с интенсивным трафиком или, скажем, запускался недостаточно часто, часть его работы можно выполнить вручную. Таблицы кэша можно очистить в любое время (выполнив команды `TRUNCATE TABLE 'cache'`, `TRUNCATE TABLE 'cache_filter'` и `TRUNCATE TABLE 'cache_page'`), после чего они будут восстановлены автоматически. Кроме того, в крайней случае можно очистить таблицы `watchdog` и `sessions`, чтобы попытаться восстановить контроль над сайтом Drupal. Обратная сторона удаления записей таблицы `watchdog` в том, что при этом утрачиваются и все сообщения об ошибках, которые могут указывать на существующие в сайте проблемы. Если требуется сохранить эти данные, перед усечением таблицы `watchdog` ее можно записать на диск. Очистка таблицы `sessions` приведет к выводу из системы текущих зарегистрированных пользователей.

Архитектуры

Архитектуры, на которых можно устанавливать Drupal — те же самые, которые подходят для другого программного обеспечения из стека LAMP, а методики, используемые для масштабирования, применимы также и к Drupal. Поэтому мы сконцентрируемся на специфичных для Drupal советах и особенностях, касающихся различных архитектур.

Единственный сервер

Это — самая простая архитектура. Веб-сервер и СУБД функционируют на одном сервере. Сервер может находиться на разделяемом или на выделенном хосте. Хотя многие небольшие сайты Drupal прекрасно выполняются на разделяемом хосте, серьезный веб-сервер, который предполагается масштабировать, должен размещаться на выделенном хосте.

При использовании односерверной архитектуры конфигурация получается простой, поскольку все делается на единственном сервере. Аналогично, связь между веб-сервером и СУБД происходит быстро, потому что нет задержек, связанных с передачей данных по сети. Очевидно, выгоднее использовать многоядерный процессор, чтобы веб-серверу и базе данных не приходилось соперничать за процессорное время.

Отдельный сервер базы данных

Если база данных является узким местом, решением проблемы может стать отдельный и мощный сервер базы данных. Часть производительности будет утеряна вследствие пересылки запросов по сети, но масштабируемость улучшится.

На заметку! Во всех ситуациях, когда приходится работать с несколькими серверами, необходимо обеспечить, чтобы они были связаны быстродействующей локальной сетью.

Отдельный сервер базы данных и кластер веб-серверов

Несколько веб-серверов обеспечивают высокую отказоустойчивость и могут обрабатывать большой объем трафика. Минимальное количество компьютеров, необходимых для создания кластера — два веб-сервера. Кроме того, должен существовать способ переключения трафика между компьютерами. Если один из компьютеров перестанет отвечать на запросы, остальная часть кластера должна быть способна справиться с возросшей нагрузкой.

Балансировка загрузки

Балансировщики загрузки распределяют сетевой трафик между веб-серверами. Существуют и другие виды балансировщиков загрузки, используемые для распределения других ресурсов, таких как жесткие диски и базы данных, но в данном случае речь идет только о распределении HTTP-запросов. В случае множества веб-серверов балансировщики загрузки позволяют веб-службам продолжать работу, если один из веб-серверов неисправен или на нем проводятся процедуры по обслуживанию.

Различают две широких категории балансировщиков загрузки. Программные балансировщики загрузки более дешевы или даже бесплатны, но текущие расходы на их обслуживание и администрирование выше, чем для аппаратных балансировщиков. Один из наиболее популярных балансировщиков загрузки для Linux — Linux Virtual Server (Виртуальный сервер Linux; www.linuxvirtualserver.org). Аппаратные балансировщики загрузки дорогостоящи, поскольку включают в себя более совершенные алгоритмы переключения серверов. Но, как правило, они надежнее программных решений.

Наряду с балансировкой загрузки конфигурация с несколькими веб-серверами сопряжена с рядом сложностей, которые в основном связаны с загрузкой файлов на сервер и поддержанием непротиворечивости кодовой базы между серверами.

Загрузки файлов на сервер и синхронизация

Когда среда Drupal функционирует на единственном веб-сервере, загруженные файлы обычно сохраняются в каталоге `files` внутри Drupal. Местоположение конфигурируется на странице Configuration ⇒ File system (Конфигурация ⇒ Файловая система). При использовании множества веб-серверов следует избегать описанного ниже сценария.

1. Пользователь загружает файл на веб-сервер А; база данных обновляется, чтобы отразить этот факт.
2. Пользователь просматривает страницу на веб-сервере В, которая ссылается на новый файл. Файл не найден!

Очевидно, необходимо сделать так, чтобы файл появился также и на веб-сервере В. Существует несколько подходов к решению этой проблемы.

Совет. Лучше всего использовать распределенную файловую систему с репликацией и высокой доступностью, такую как GlusterFS или AndrewFS. Система Rsync полностью выходит из-под контроля при использовании более двух веб-серверов, а NFS не обеспечивает высокого уровня доступности, поэтому обе эти файловые системы вряд ли можно рекомендовать для реальных рабочих сайтов.

Использование разделяемой, монтируемой файловой системы

Вместо синхронизации множества веб-серверов можно развернуть разделяемую, монтируемую файловую систему, которая хранит файлы в единственном месте на файловом сервере. Веб-серверы затем могут монтировать файловый сервер, используя протокол типа NFS (Network File System — сетевая файловая система). Преимущество этого подхода состоит в том, что при необходимости можно легко добавить недорогие дополнительные веб-серверы, а ресурсы могут быть сконцентрированы на мощном файловом сервере с избыточной системой запоминающих устройств вроде RAID 5. Основным недостатком такой системы связан с наличием единственной точки отказа; если сервер или смонтированная файловая система выйдет из строя, это коснется и сайта, если только не будет создан кластер файловых серверов.

Если требуется обслуживать много крупных медиа-файлов, возможно, лучше делать это на отдельном сервере с применением облегченного веб-сервера типа `lighttpd`. Это позволит избежать наличия на веб-серверах множества длительно выполняемых процессов, которые конкурируют с запросами, обрабатываемыми Drupal. Простое решение предусматривает использование на веб-сервере правила перезаписи, что позволит переадресовывать все входящие запросы файлов определенного типа на статический сервер. Ниже показан пример правила перезаписи для Apache, которое перезаписывает все запросы JPEG-файлов:

```
RewriteCond %{REQUEST_URI} ^/(.*\.jpg)$ [NC]
RewriteRule .* http://static.example.com/%1 [R]
```

Недостаток описанного подхода в том, что веб-серверы по-прежнему выполняют дополнительную работу по переадресации трафика на файловый сервер. Более совершенное решение состоит в том, чтобы перезаписать все файловые URL внутри Drupal, тогда веб-серверы больше не будут вовлечены в обработку запросов статических файлов. Однако в настоящее время не существует простого способа внести такое изменение в ядро Drupal.

За рамками единственной файловой системы

Если требуемый для хранения объем начинает превышать возможности единственной файловой системы, не исключено, что придется программно реализовать некоторую абстракцию хранения. Один из вариантов — воспользоваться внешней запоминающей системой, такой как служба S3 сайта Amazon.

Множество серверов базы данных

При использовании множества серверов для базы данных возникают дополнительные сложности, потому что вставляемые и обновляемые данные должны реплицироваться или секционироваться между серверами.

Репликация баз данных

При репликации баз данных в MySQL все записи делаются в единственную главную базу данных. Затем они реплицируются в одну или несколько подчиненных баз данных. Операции чтения могут осуществляться как из главной, так и из любой подчиненной базы данных. В многоуровневой архитектуре подчиненные базы данных также могут быть главными.

Секционирование баз данных

Поскольку среда Drupal способна обрабатывать множество подключений к базам данных, другая стратегия масштабирования архитектуры баз данных предполагает помещение некоторых таблиц в одну базу данных на одном компьютере, а остальных таблиц — в другую базу данных на другом компьютере. Например, перемещение всех таблиц кэша в отдельную базу данных на отдельном компьютере и создание псевдонимов для всех запросов к этим таблицам с использованием механизма табличных префиксов Drupal может помочь масштабированию сайта.

Обнаружение узкого места

Если сайт Drupal не работает с той скоростью, с которой ожидалось, первый шаг должен состоять в анализе источника проблем. Они могли возникнуть на веб-сервере, в операционной системе, в базе данных, в файловой системе и в сети.

Знание того, как оценивать производительность и масштабируемость системы, позволяет быстро изолировать и ликвидировать ее узкие места даже во время кризисной загрузки. Обнаружить узкие места можно с помощью нескольких простых инструментов, по ходу дела находя ответы на возникающие вопросы. Ниже представлен один из подходов по выявлению проблем с низкой производительностью сервера. Начинать следует с понимания того, что производительность может ограничиваться одной из следующих составляющих: центральный процессор, оперативная память, система ввода-вывода, ширина полосы пропускания сети. Давайте начнем с выяснения ответов на перечисленные ниже вопросы.

- *Полностью ли используется центральный процессор?* Если исследование степени использования центрального процессора с помощью утилиты `top` в Unix или диспетчера задач в Windows дает результат, близкий к 100%, следует выяснить, что конкретно вызывает такую загрузку. Просмотр списка процессов позволит выяснить, что именно загружает процессор — веб-сервер или база данных. Обе эти проблемы разрешимы.
- *Насколько интенсивно происходит страничный обмен на сервере?* Если серверу не хватает физической памяти для выполнения назначенной ему задачи, операционная система будет использовать виртуальную память (диск), чтобы справиться с нагрузкой. Чтение и запись с диска осуществляется значительно медленнее чтения и записи в физической памяти. Если сервер интенсивно использует запись страниц памяти на диск, нужно выяснить, чем это вызвано.
- *Полностью ли используются диски?* Если исследование дисковой подсистемы с помощью инструмента типа `vmstat` в Unix или монитора производительности в Windows показывает, что дисковая активность не удовлетворяет требованиям операционной системы, несмотря на наличие достаточного объема свободной памяти, это значит, что существует проблема ввода-вывода. Причиной может быть чрезмерно подробная регистрация, неверно сконфигурированная база данных, которая создает множество временных таблиц на диске, фоновое выполнение сценария, использование неподходящего уровня RAID для приложения с частыми операциями записи и т.д.
- *Не перегружена ли линия связи?* Если сетевой канал полностью заполнен, существует только два решения. Первое — обзавестись линией связи с большей пропускной способностью. Второе — передавать меньшие объемы информации, обеспечивая при этом необходимое сжатие данных.

Совет. Исследование производительности обслуживания ваших страниц извне сервера также может оказаться полезным. Инструмент, подобный YSlow (<http://developer.yahoo.com/yslow/help/>), может помочь в поисках причины того, почему страницы загружаются не так быстро, как хотелось бы, хотя потолок возможностей процессора, памяти или системы ввода-вывода еще не достигнут. По адресу <http://wimleers.com/article/improving-drupals-page-loading-performance> доступна полезная статья, посвященная YSlow и Drupal.

Перегрузка центрального процессора веб-сервера

Если возможности центрального процессора полностью исчерпаны, и список процессов показывает, что ресурсы потребляет в основном веб-сервер, а не СУБД (о ней речь пойдет позже), нужно изучить возможность сокращения накладных расходов веб-сервера при обслуживании запросов. Часто причиной перегрузки оказываются выполнение PHP-кода. Решение этой проблемы было подробно описано в разделе “Оптимизация PHP” ранее в главе.

Зачастую специализированный код и модули, которые демонстрировали хорошую производительность на небольшом сайте, становятся узким местом после перемещения на рабочий сайт. Циклы, интенсивно загружающие центральный процессор, алгоритмы, потребляющие память, и операции извлечения из баз данных крупных массивов информации могут быть идентифицированы путем профилирования кода. При этом можно определить, на что PHP тратит большую часть времени и, следовательно, где придется затратить большую часть времени на отладку.

Если даже после добавления кэша кодов операций и оптимизации кода веб-сервер не может справиться с нагрузкой, значит, пришло время сменить системный блок на блок с более быстродействующим процессором (или несколькими процессорами) либо переходить на архитектуру с несколькими веб-серверами.

Нехватка оперативной памяти на веб-сервере

Отпечаток оперативной памяти процесса веб-сервера, обслуживающего запрос, включает все модули, загруженные веб-сервером (такие как модули `mod_mime`, `mod_rewrite` Apache и т.д.), а также память, используемую интерпретатором PHP. Чем больше модулей веб-сервера и Drupal активировано, тем больше оперативной памяти расходуется на запрос.

Поскольку оперативная память — ограниченный ресурс, необходимо определить, сколько ее используется на каждый запрос, и для обработки какого количества запросов сконфигурирован веб-сервер. Чтобы понять, сколько реальной оперативной памяти применяется в среднем для каждого запроса, воспользуйтесь такой утилитой, как `top` (в Linux), которая выводит список процессов. На сервере Apache максимальное количество одновременно обслуживаемых запросов устанавливается с помощью директивы `MaxClients`. Распространенная ошибка — полагать, что решение проблемы перегруженного веб-сервера сводится к увеличению значения `MaxClients`. Это только усугубит проблему, поскольку придется одновременно обслуживать еще больше запросов. В конечном итоге оперативная память будет исчерпана, сервер начнет свопинг и перестанет реагировать на запросы. Предположим, например, что на веб-сервере установлено 2 Гбайт оперативной памяти, а каждый запрос к Apache требует примерно 20 Мбайт (фактическое значение можно проверить с помощью `top` в Linux или диспетчера задач в Windows). Подходящее значение для `MaxClients` можно вычислить по приведенной ниже формуле; при этом не следует забывать, что нужно зарезервировать память как для самой операционной системы, так и для других важных процессов:

$$2 \text{ Гбайт ОЗУ} / 20 \text{ Мбайт на процесс} = 100 \text{ MaxClients}$$

Если на сервере постоянно ощущается нехватка оперативной памяти даже после отключения ненужных модулей и профилирования всех специализированных модулей или кода, на следующем шаге необходимо удостовериться, что СУБД и операционная система не являются узким местом. Если это так, увеличьте объем оперативной памяти. Если причина не в СУБД и не в операционной системе, значит, на сервер просто поступает больше запросов, чем он может обслужить; решение состоит в добавлении веб-серверов.

Совет. Поскольку использование памяти процессами Apache имеет тенденцию увеличиваться до уровня самой требовательной к памяти страницы, обслуживаемой данным дочерним процессом, память можно освободить, установив низкое значение для `MaxRequestsPerChild` — например, 300 (конкретное число будет зависеть от фактической ситуации). Работа сервера Apache несколько замедлится, поскольку ему придется генерировать новые дочерние процессы, однако новые дочерние процессы будут потреблять меньше оперативной памяти, что позволит обслуживать больше запросов при меньшем объеме памяти. Настройка по умолчанию для `MaxRequestsPerChild` — значение 0, означающее, что дочерние процессы никогда не будут иметь конечного срока действия.

Идентификация ресурсоемких запросов базы данных

Если требуется получить представление о том, что происходит, когда генерируется та или иная страница, в этом поможет модуль `devel.module`. Он имеет опцию отображения всех запросов, которые необходимы для генерации страницы, а также времени выполнения каждого запроса.

Другой способ выяснения того, какие запросы выполняются слишком долго — включение регистрации медленных запросов в MySQL. Это делается в файле параметров MySQL (`my.cnf`) следующим образом:

```
# The MySQL server
[mysqld]
log-slow-queries
```

Это приведет к регистрации всех запросов, для выполнения которых требуется больше десяти секунд, в журнальном файле `example.com-slow.log` каталога данных MySQL. Ниже показано, как изменить количество секунд и местоположение журнального файла; в этом примере порог медленного запроса устанавливается равным пяти секундам, а в качестве имени файла выбирается `example-slow.log`:

```
# The MySQL server
[mysqld]
long_query_time = 5
log-slow-queries = /var/log/mysql/example-slow.log
```

Идентификация ресурсоемких страниц

Чтобы узнать, какие страницы являются самыми ресурсоемкими, активируйте модуль статистики (`statistics`) в Drupal. Хотя модуль `statistics` увеличивает нагрузку на сервер (поскольку фиксирует в базе данных статистику доступа к сайту), он полезен для выяснения того, какие страницы просматриваются чаще других и таким образом являются кандидатами на оптимизацию запросов. Модуль также отслеживает полное время генерации страницы за период, который можно указать на странице Configuration ⇒ Statistics (Конфигурация ⇒ Статистика). Это полезно для идентификации неуправляемых поисковых роботов, которые поглощают ресурсы системы. После этого им можно запретить посещения сайта, перейдя на страницу Reports ⇒ Top visitors (Отчеты ⇒ Основные посетители) и щелкнув на ссылке ban (запретить). Но будьте внимательны — очень просто запретить посещения “добропорядочному” поисковому роботу, который направляет трафик на ваш сайт. Прежде чем запретить посещение поисковому роботу, вначале обязательно исследуйте его происхождение.

Идентификация ресурсоемкого кода

Ниже показан код, чрезмерно расходующий ресурсы:

```
// Очень дорогостоящий и неудачный способ получения заголовков нод.
// Сначала извлекаются идентификаторы всех опубликованных нод.
```

```
$query = db_select('node', 'n');
$query->fields('n', array('nid'));
$query->condition("n.status", 1);
$query->addTag('node_access');
$result = $query->execute();
```

```
// Теперь выполнить node_load() для каждой отдельной ноды
// и сохранить ее заголовок.
```

```
foreach($result as $row) {
    $node = node_load($row->nid);
    $titles[] = check_plain($node->title);
}
```

Полная загрузка ноды — дорогостоящая операция: запускаются хуки, модули выполняют запросы к базе данных, чтобы добавить или изменить ноду, а память расходуется для кэширования ноды во внутреннем кэше `node_load()`. Если зависимость от модификации ноды модулем отсутствует, намного быстрее непосредственно выполнить собственный запрос к таблице нод. Разумеется, этот пример несколько надуман, однако аналогичную картину можно часто наблюдать, когда данные извлекаются с помощью множества запросов, которые можно было бы объединить в один, или когда производится ненужная загрузка ноды.

Совет. В Drupal имеется внутренний механизм кэширования (использующий статическую переменную), который активизируется, когда нода загружается более одного раза на запрос. Например, после вызова `node_load(1)` нода с номером 1 полностью загружается и кэшируется. Если в том же запросе происходит второй вызов `node_load(1)`, Drupal возвратит кэшированный результат для предварительно загруженной ноды, имеющей тот же самый идентификатор.

Оптимизация таблиц

Замедление SQL-запросов может объясняться неудачной реализацией таблиц SQL в дополнительных модулях. Например, к медленному выполнению запросов могут привести столбцы без индексов. Быстрый способ выяснения того, как запрос выполняется в MySQL — предварение запроса, протоколированного в журнале медленных запросов, словом EXPLAIN и его передача MySQL. Результатом будет таблица, показывающая, какие индексы были использованы. За более подробной информацией обратитесь к специализированной книге по MySQL.

Кэширование запросов вручную

Если от ресурсоемких запросов избавиться не удастся, возможно, имеет смысл вручную кэшировать результаты вашим модулем. API-интерфейс кэширования Drupal подробно описан в главе 16.

Изменение типа таблицы с MyISAM на InnoDB

Двумя обычно используемыми вариантами механизмов хранения MySQL, которые часто называются *типами таблиц*, являются MyISAM и InnoDB. По умолчанию Drupal применяет InnoDB.

В MyISAM используется блокировка на уровне таблиц, в то время как в InnoDB — блокировка на уровне строк. Блокировка (locking) важна для сохранения целостности базы данных; она препятствует одновременному обновлению одних и тех же данных двумя разными процессами базы данных. На практике различие в стратегиях блокировки означает, что в MyISAM при выполнении записи блокируется вся таблица. Поэтому при добавлении нового комментария на сайте Drupal с высокой посещаемостью чтение всех существующих комментариев блокируются. В InnoDB такой проблемы нет, поскольку блокируется только строка (или строки), записываемая в данный момент; другие серверные потоки могут продолжать работать с остальными строками. Однако в случае MyISAM чтение таблицы осуществляется быстрее, а инструменты обслуживания и восстановления более совершенны.

Дополнительные сведения по архитектурам хранения таблиц MySQL можно найти по адресам:

http://dev.mysql.com/tech-resources/articles/storage-engine/part_1.html и
<http://dev.mysql.com/doc/refman/5.1/en/storage-engines.html>

Чтобы проверить, не является ли блокировка таблиц причиной низкой производительности, проанализируйте состязания за блокировки, проверив переменные состояния `Table_locks_immediate` и `Table_locks_waited` в MySQL:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 1151552 |
| Table_locks_waited | 15324 |
+-----+-----+
```

Значение `Table_locks_immediate` — это количество случаев, когда блокировка таблицы была произведена немедленно, а `Table_locks_waited` — количество раз, когда блокировка таблицы сопровождалось ожиданием. Если значение `Table_locks_waited` велико, и имеются проблемы с производительностью, можно попробовать разделить большие таблицы на части. Например, можно предусмотреть отдельную таблицу кэша для специализированного модуля либо поискать возможность уменьшения размеров таблиц или частоты применения команд блокировки таблиц. Один из способов уменьшения размеров таких таблиц, как `cache_*`, `watchdog` и `accesslog`, состоит в уменьшении времени жизни данных. Это можно выполнить через административный интерфейс Drupal. Кроме того, удостоверьтесь в том, что `cron` выполняется не реже, чем раз в час; это обеспечит своевременное усечение таких таблиц.

Поскольку Drupal может использоваться множеством различных способов, невозможно дать всеобъемлющие рекомендации относительно того, какой механизм должны применять те или иные таблицы. Тем не менее, в общем случае хорошими кандидатами на преобразование в InnoDB являются таблицы `cache`, `watchdog`, `sessions` и `accesslog`. К счастью, преобразование в InnoDB осуществляется очень просто:

```
ALTER TABLE accesslog TYPE='InnoDB';
```

Разумеется, это преобразование должно проводиться, когда сайт находится в автономном режиме, резервная копия существующих данных создана, и вы четко представляете себе характеристики таблиц InnoDB.

Для настройки производительности MySQL обратитесь к сценарию настройки производительности из www.day32.com/MySQL/, который предоставляет рекомендации по настройке переменных сервера MySQL.

Резюме

После прочтения данной главы вы должны уметь выполнять следующие действия.

- Уметь выявлять узкие места, снижающие производительность.
- Знать, как оптимизировать веб-сервер.
- Уметь оптимизировать базу данных.
- Знать специфичные для Drupal методы оптимизации.
- Знать возможные варианты архитектур с несколькими серверами.

ГЛАВА 24

Инсталляционные профили

При первой установке Drupal 7 вам предлагалось выбрать инсталляционный профиль, который должен использоваться в качестве начальной конфигурации нового сайта Drupal. При этом предлагались варианты *Standard* (Стандартный) и *Minimal* (Минимальный). Каждая опция представляет заранее определенный подход к установке Drupal, в том числе включаемые модули, создаваемые типы контента, выбираемые и включаемые темы, а также включаемые и присваиваемые конкретным областям блоки. Инсталляционный профиль *Standard* предоставляет сравнительно полный сайт Drupal со многими из функций, которые, скорее всего, будут использоваться на обычном сайте Drupal. Инсталляционный профиль *Minimal* реализует минимальный набор функций и компонентов Drupal, достаточный, чтобы привести сайт в рабочее состояние.

Создание инсталляционного профиля — относительно легкий процесс; фактически, профиль установки — это всего лишь модуль Drupal. Если вы можете создать модуль, то можете создать и инсталляционный профиль. Из инсталляционных профилей можно выполнять все те действия, которые можно производить с помощью модулей, в том числе использовать полный интерфейс Drupal API и создавать функции обновления для перехода от одной версии к другой. В этой главе вы научитесь создавать собственный инсталляционный профиль, который строит сайт Drupal с функциями и компонентами, необходимыми для удовлетворения конкретных требований.

Создание нового инсталляционного профиля

Создание нового инсталляционного профиля целесообразно начинать с исследования профиля *Standard* Drupal 7. Профиль *Standard* содержит многие из тех функций, которые будут требоваться для большинства сайтов, и предоставляет понятную и удобную платформу для добавления новых средств и функциональности. В качестве примера будет создан новый профиль *enhanced* и расширены функции и возможности, определенные в стандартном профиле Drupal.

Для начала создадим подкаталог *enhanced* внутри каталога `/profiles`, расположенного в базовом каталоге установки Drupal. Поскольку планируется расширить функции функциональные возможности стандартного инсталляционного профиля, содержимое каталога *standard* будет скопировано в каталог *enhanced*.

Три файла, которые нужно будет скопировать, описаны в табл. 24.1.

Таблица 24.1. Файлы, необходимые для инсталляционного профиля: замените `profilename` фактическим именем своего профиля

Имя файла	Описание
<code>profilename.info</code>	Содержит основную информацию об инсталляционном профиле. Открыв этот файл, легко убедиться, что его структура и содержимое идентично файлу <code>.info</code> стандартного модуля Drupal
<code>profilename.install</code>	Описывает ключевые особенности и атрибуты нового экземпляра Drupal, такие как фильтры, блоки, типы контента, словари таксономии и другие атрибуты
<code>profilename.profile</code>	Используется для модифицирования формы инсталляционного профиля за счет вставки дополнительных полей, необходимых для поддержки нового инсталляционного профиля

Прежде чем приступить к созданию нового усовершенствованного инсталляционного профиля, переименуем эти файлы соответственно в `enhanced.info`, `enhanced.install` и `enhanced.profile`.

Файл `enhanced.info`

Этот файл содержит основную информацию, необходимую ядру Drupal для идентификации и определения ключевых атрибутов инсталляционного профиля, в том числе:

- имя инсталляционного профиля;
- номер версии, связанной с данным файлом;
- версия ядра Drupal, поддерживаемого этим инсталляционным профилем;
- список зависимостей (модулей), требуемых нашим профилем — дополнительные модули, выходящие за рамки активируемых в стандартном профиле, добавлены в конце этого списка;
- имя файла `.profile` данного профиля.

Поскольку мы начали со стандартного профиля, нужно обновить имя, описание и атрибуты файлов, чтобы они отражали особенности инсталляционного профиля `enhanced`. Также понадобится изменить список зависимых модулей, чтобы удовлетворить любым специфичным требованиям нового инсталляционного профиля. Полный файл `.info` модуля `enhanced` представлен ниже.

```
name = Enhanced
description = An enhanced profile.
core = 7.x
dependencies[] = block
dependencies[] = color
dependencies[] = comment
dependencies[] = contextual
dependencies[] = dashboard
dependencies[] = help
dependencies[] = image
dependencies[] = menu
dependencies[] = path
dependencies[] = taxonomy
dependencies[] = dblog
dependencies[] = search
dependencies[] = shortcut
dependencies[] = toolbar
```

```
dependencies[] = overlay
dependencies[] = field_ui
dependencies[] = file
dependencies[] = rdf
// Добавления к стандартному профилю.
dependencies[] = forum
dependencies[] = blog
dependencies[] = poll
dependencies[] = book
files[] = enhanced.profile
```

Файл `enhanced.profile`

Этот файл запускает `hook_form_formname_alter` и в этом примере устанавливает поле имени сайта на форме в имя сайта, определенное сервером. Можно начать с пустого файла `.profile` и добавлять в него записи, необходимые для поддержки предъявляемых требований; однако ни одна запись не является обязательной, поэтому пустой файл `.profile` — это все, что требуется для успешного создания профиля.

```
/**
 * Реализует hook_form_alter().
 *
 * Позволяет профилю изменять форму конфигурации сайта.
 */
function enhanced_form_install_configure_form_alter(&$form, $form_state, $form_id)
{
  if ($form_id == 'install_configure_form') {
    // Устанавливает для поля имени сайта значение по умолчанию.
    $form['site_information']['site_name']['#default_value'] =
      $_SERVER['SERVER_NAME'];
  }
}
```

Файл `enhanced.install`

Именно в этом файле определяются и устанавливаются все средства, функции, переменные и параметры конфигурации. В файле `.install` мы определим следующее:

- входные форматы, которые будут определены и включены;
- блоки, которые будут включены и назначены областям;
- типы контента, которые будут созданы и включены;
- поля, которые будут созданы и связаны с типами контента;
- полномочия, которые будут выданы администраторам сайта;
- роли, которые будут созданы;
- тема, которая будет определена в качестве активной.

Все настройки помещаются в функцию хука инсталляции. Применительно к усовершенствованному инсталляционному профилю имя функции приобретает вид `enhanced_install()`, как показано в следующем примере.

```
/**
 * Реализация hook_install().
 *
 * Выполнение действий по настройке сайта для этого профиля.
 */
function enhanced_install() {
```

Прежде всего, необходимо установить текстовые форматы, которые будут использоваться в рамках сайта. В качестве входных форматов мы будем применять фильтрованный HTML и полный HTML, которые являются стандартными в большинстве сайтов Drupal. Чтобы определить входной формат, ему нужно назначить имя, присвоить вес, установить его состояние в активное (1) и определить список фильтров, которые включены во входной формат. В следующем примере демонстрируется установка фильтрованного HTML как входного формата. Этот входной формат применяет URL-фильтр (заменяет ссылки URL-адресами), HTML-фильтр (удаляет нежелательные дескрипторы), фильтр разрыва строк (заменяет символы возврата каретки дескрипторами `<br/>`) и фильтры HTML-корректора. Каждому фильтру в списке назначается вес, который определяет порядок применения фильтров к контенту.

```
// Добавление текстовых форматов.
$filtered_html_format = array(
  'name' => 'Filtered HTML',
  'weight' => 0,
  'filters' => array(
    // URL-фильтр.
    'filter_url' => array(
      'weight' => 0,
      'status' => 1,
    ),
    // HTML-фильтр.
    'filter_html' => array(
      'weight' => 1,
      'status' => 1,
    ),
    // Фильтр разрыва строк.
    'filter_autop' => array(
      'weight' => 2,
      'status' => 1,
    ),
    // Фильтр HTML-корректора.
    'filter_htmlcorrector' => array(
      'weight' => 10,
      'status' => 1,
    ),
  ),
);
```

Следующий шаг, который должен быть предпринят в файле `.install` — преобразование ранее созданной структуры в объект и сохранение входного формата посредством API-вызова `filter_format_save`:

```
$filtered_html_format = (object) $filtered_html_format;
filter_format_save($filtered_html_format);
```

Теперь определим входной формат полного HTML. По своей структуре этот входной формат идентичен фильтрованному HTML, за исключением того, что он не содержит HTML-фильтра, удаляющего нежелательные HTML-дескрипторы. Последовательность действий остается прежней — назначение имени, веса и определение списка фильтров, которые будут включены в этот входной формат:

```
$full_html_format = array(
  'name' => 'Full HTML',
  'weight' => 1,
  'filters' => array(
```

```

// URL-фильтр.
'filter_url' => array(
  'weight' => 0,
  'status' => 1,
),
// Фильтр разрыва строк.
'filter_autop' => array(
  'weight' => 1,
  'status' => 1,
),
// Фильтр HTML-корректора.
'filter_htmlcorrector' => array(
  'weight' => 10,
  'status' => 1,
),
),
);

```

Следующий шаг — преобразование только что созданной структуры в объект и последующее сохранение входного формата в базе данных Drupal посредством API-вызова `filter_format_save`:

```

$full_html_format = (object) $full_html_format;
filter_format_save($full_html_format);

```

Следующий шаг процесса — определение блоков, которые будут включены и назначены областям данной темы. Основные атрибуты, присвоенные определению каждого блока, описаны в табл. 24.2.

Таблица 24.2. Атрибуты блоков, используемые для определения каждого блока, который будет активирован во время процесса установки

Атрибут	Описание
Module	Имя модуля, в котором определен блок
Delta	Имя блока, определенное в модуле (для выяснения списка блоков, определенных внутри модуля, найдите <code>hook_block_info</code>)
Theme	Имя темы, в которой блок будет присвоен
Status	Определяет, включен (1) или отключен (0) блок
Region	Определение области в теме, в которой должен отображаться блок
Cache	Определяет, кэшируется блок или нет (-1)

Следующий код отображает список блоков, которые автоматически активируются и назначаются областям усовершенствованным инсталляционным профилем:

```

// Активация ряда стандартных блоков.
$values = array(
  array(
    'module' => 'system',
    'delta' => 'main',
    'theme' => 'garland',
    'status' => 1,
    'weight' => 0,
    'region' => 'content',
    'pages' => '',

```

```

    'cache' => -1,
  ),
  array(
    'module' => 'search',
    'delta' => 'form',
    'theme' => 'garland',
    'status' => 1,
    'weight' => -1,
    'region' => 'sidebar_first',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'node',
    'delta' => 'recent',
    'theme' => 'seven',
    'status' => 1,
    'weight' => 10,
    'region' => 'dashboard_main',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'user',
    'delta' => 'login',
    'theme' => 'garland',
    'status' => 1,
    'weight' => 0,
    'region' => 'sidebar_first',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'system',
    'delta' => 'navigation',
    'theme' => 'garland',
    'status' => 1,
    'weight' => 0,
    'region' => 'sidebar_first',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'system',
    'delta' => 'management',
    'theme' => 'garland',
    'status' => 1,
    'weight' => 1,
    'region' => 'sidebar_first',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'system',
    'delta' => 'powered-by',
    'theme' => 'garland',

```

```

    'status' => 1,
    'weight' => 10,
    'region' => 'footer',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'system',
    'delta' => 'help',
    'theme' => 'garland',
    'status' => 1,
    'weight' => 0,
    'region' => 'help',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'system',
    'delta' => 'main',
    'theme' => 'seven',
    'status' => 1,
    'weight' => 0,
    'region' => 'content',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'system',
    'delta' => 'help',
    'theme' => 'seven',
    'status' => 1,
    'weight' => 0,
    'region' => 'help',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'user',
    'delta' => 'login',
    'theme' => 'seven',
    'status' => 1,
    'weight' => 10,
    'region' => 'content',
    'pages' => '',
    'cache' => -1,
  ),
  array(
    'module' => 'user',
    'delta' => 'new',
    'theme' => 'seven',
    'status' => 1,
    'weight' => 0,
    'region' => 'dashboard_sidebar',
    'pages' => '',
    'cache' => -1,
  ),
),

```



```

array(
  'module' => 'search',
  'delta' => 'form',
  'theme' => 'seven',
  'status' => 1,
  'weight' => -10,
  'region' => 'dashboard_sidebar',
  'pages' => '',
  'cache' => -1,
),
// Дополнительные блоки помимо определенных в стандартном профиле.
array(
  'module' => 'blog',
  'delta' => 'recent',
  'theme' => 'garland',
  'status' => 1,
  'weight' => 5,
  'region' => 'sidebar_first',
  'pages' => '',
  'cache' => -1,
),
array(
  'module' => 'forum',
  'delta' => 'active',
  'theme' => 'garland',
  'status' => 1,
  'weight' => 10,
  'region' => 'sidebar_first',
  'pages' => '',
  'cache' => -1,
),
array(
  'module' => 'forum',
  'delta' => 'new',
  'theme' => 'garland',
  'status' => 1,
  'weight' => 10,
  'region' => 'sidebar_first',
  'pages' => '',
  'cache' => -1,
),
array(
  'module' => 'poll',
  'delta' => 'recent',
  'theme' => 'garland',
  'status' => 1,
  'weight' => 15,
  'region' => 'sidebar_first',
  'pages' => '',
  'cache' => -1,
),
);

```

Следующий шаг в процессе — сохранение информации о конфигурации блоков в таблице `block` базы данных Drupal. Следующий код вставляет все только что определенные блоки в эту таблицу:

470 Глава 24. Инсталляционные профили

```
$query = db_insert('block')->fields(array('module', 'delta', 'theme',  
'status', 'weight',  
'region', 'pages', 'cache'));  
foreach ($values as $record) {  
    $query->values($record);  
}  
$query->execute();
```

Следующий шаг в процессе установки — определение типов контента, которые будут создаваться и включаться во время процесса установки. Основные атрибуты, связанные с типом контента, перечислены в табл. 24.3.

Таблица 24.3. Атрибуты типа контента

Атрибут	Описание
Type	Внутреннее имя типа контента
Name	Имя типа контента, которое отображается на административных страницах
Base	Основополагающий тип контента, который используется для создания нового типа контента
Description	Описание типа контента, которое отображается на административных страницах
Custom	Определяет, является ли тип контента нестандартным типом (1)
Modified	Определяет, модифицировался ли тип контента; т.к. мы только создаем тип контента, значение устанавливается в 1 (модифицировался)
Locked	Определяет, может ли тип контента модифицироваться (0) или не может (1)

Для инсталляционного профиля `enhanced` мы воспользуемся типами контента `Basic page` (Базовая страница) и `Article` (Статья), определенными в стандартном инсталляционном профиле, и создадим новый тип контента, который будет применяться только для контента, связанного с новостями:

```
$types = array(  
    array(  
        'type' => 'page',  
        'name' => st('Basic page'),  
        'base' => 'node_content',  
        'description' => st("Use <em>basic pages</em> for your static content,  
such as an 'About us' page."),  
        'custom' => 1,  
        'modified' => 1,  
        'locked' => 0,  
    ),  
    array(  
        'type' => 'article',  
        'name' => st('Article'),  
        'base' => 'node_content',  
        'description' => st('Use <em>articles</em> for content that requires an  
image and tags.'),  
        'custom' => 1,  
        'modified' => 1,  
        'locked' => 0,  
    ),  
);
```

```
// Новый тип контента, добавленный для инсталляционного профиля enhanced.
array(
  'type' => 'news',
  'name' => st('News'),
  'base' => 'node_content',
  'description' => st('Use <em>news</em> for news related content.'),
  'custom' => 1,
  'modified' => 1,
  'locked' => 0,
),
);
```

Следующий шаг в процессе — сохранение типов контента посредством API-вызовов `node_type_set_defaults`, `node_type_save` и `node_add_body_field` для каждого из определенных ранее типов контента:

```
foreach ($types as $type) {
  $type = node_type_set_defaults($type);
  node_type_save($type);
  node_add_body_field($type);
}
```

После создания типов контента следующим шагом нужно установить RDF-отображения для каждого из типов контента, определенных на предыдущем шаге. Стандартный инсталляционный профиль определил отображения для типов контента страницы и статьи. Мы добавим отображение типа контента новостей.

```
$rdf_mappings = array(
  array(
    'type' => 'node',
    'bundle' => 'page',
    'mapping' => array(
      'rdftype' => array('foaf:Document'),
    ),
  ),
  array(
    'type' => 'node',
    'bundle' => 'article',
    'mapping' => array(
      'rdftype' => array('sioc:Item', 'foaf:Document'),
      'field_image' => array(
        'predicates' => array('rdfs:seeAlso'),
        'type' => 'rel',
      ),
      'field_tags' => array(
        'predicates' => array('dc:subject'),
        'type' => 'rel',
      ),
    ),
  ),
);
// Следующий код был добавлен для инсталляционного профиля enhanced.
array(
  'type' => 'node',
  'bundle' => 'news',
  'mapping' => array(
    'rdftype' => array('foaf:Document'),
  ),
),
);
```

После того, как RDF-отображения определены, их нужно сохранить в базе данных, что выполняется посредством API-вызова `rdf_mapping_save` для каждого из отображений:

```
foreach ($rdf_mappings as $rdf_mapping) {
  rdf_mapping_save($rdf_mapping);
}
```

После определения типов контента понадобится установить несколько атрибутов типов контента. Стандартный инсталляционный профиль устанавливает атрибуты типа `Basic page` так, чтобы по умолчанию базовые страницы не продвигались на переднюю страницу, а комментарии были отключены. Для типа контента новостей желательно, чтобы эти страницы продвигались на переднюю страницу и чтобы их посетители могли отправлять свои комментарии, поэтому мы оставим значения атрибутов, установленные по умолчанию:

```
variable_set('node_options_page', array('status')); // не продвигать
// базовые страницы на домашнюю страницу
variable_set('comment_page', COMMENT_NODE_HIDDEN); // запретить
// комментирование на базовых страницах
```

Следующий атрибут, устанавливаемый стандартным инсталляционным профилем, определяет, отображаются ли имя автора и предоставленные значения даты и времени на экране при визуализации ноды на странице. Поскольку нежелательно, чтобы эта информация отображалась для типов контента `Basic page`, мы установим атрибут, определяющий вывод имени автора и даты, в значение `false`:

```
variable_set('node_submitted_page', FALSE);
```

Следующий набор параметров определяет, могут ли пользователи загружать картинку для своего профиля и отображаются ли эти картинки с их сообщениями и комментариями. Следующий набор параметров конфигурации устанавливает, разрешены ли картинки, и атрибуты, которые определяют, как эти изображения выводятся на сайте:

```
variable_set('user_pictures', '1'); // устанавливает атрибут так, чтобы
// пользовательские изображения были разрешены (1)
variable_set('user_picture_dimensions', '1024x1024'); // устанавливает
// максимальные размеры изображения
variable_set('user_picture_dimensions', '800'); // устанавливает
// максимальный размер файла изображения
variable_set('user_picture_style', 'thumbnail'); // устанавливает размер
// по умолчанию, который будет визуализироваться на странице
```

Теперь определим способ обработки создания пользовательской учетной записи. Есть три возможных значения, которые можно использовать для определения способа обработки пользовательских учетных записей в системе, как описано в табл. 24.4.

Мы используем подход, при котором пользователи могут зарегистрировать учетную запись, но администратор должен утвердить и активизировать ее:

```
variable_set('user_register', USER_REGISTER_VISITORS_ADMINISTRATIVE_APPROVAL);
```

Следующий шаг в процессе установки — настройка таксономии. Мы создадим словарь `Tags` (Дескрипторы), который позволит авторам свободно помечать контент.

Таблица 24.4. Параметры создания учетной записи пользователя

Значение	Описание
USER_REGISTER_ADMINISTRATORS_ONLY	Создание учетных записей на сайте разрешено только администраторам
USER_REGISTER_VISITORS_ADMINISTRATIVE_APPROVAL	Посетители могут создать учетную запись, но эта учетная запись не будет активной, пока администратор не утвердит и не активизирует ее
USER_REGISTER_VISITORS	Посетители могут создать учетную запись, и эта учетная запись автоматически утверждается и активизируется

Сначала определим описание и справочный текст, связанный со словарем, а затем создадим объект словаря:

```
$description = st('Use tags to group articles on similar topics into categories.');
```

```
$help = st('Enter a comma-separated list of words to describe your content.');
```

```
$vocabulary = (object) array(
    'name' => 'Tags', // имя словаря
    'description' => $description,
    'machine_name' => 'tags',
    'help' => $help,
);
```

После определения объекта словаря сохраним его в базе данных Drupal с помощью API-вызова `taxonomy_vocabulary_save`:

```
taxonomy_vocabulary_save($vocabulary);
```

Затем создадим поле, используя словарь `Tags`, и присвоим поле типу контента статьи:

```
$field = array(
    // Создать имя поля с использованием имени компьютера
    // только что созданного словаря.
    'field_name' => 'field_' . $vocabulary->machine_name,

    // Определить тип поля как ссылку на термин таксономии.
    'type' => 'taxonomy_term_reference',

    // Установить неограниченное количество элементов для пометки.
    'cardinality' => FIELD_CARDINALITY_UNLIMITED,

    // Установить список разрешенных значений в список терминов в словаре.
    'settings' => array(
        'allowed_values' => array(
            array(
                'vid' => $vocabulary->vid,
                'parent' => 0,
            ),
        ),
    ),
);

// Создать поле с помощью API-вызова field_create_field.
field_create_field($field);
```

После того как поле создано, его можно присвоить типу контента статьи:

```
$instance = array(
  // Использовать только что созданное поле.
  'field_name' => 'field_' . $vocabulary->machine_name,
  // Назначить его ноду.
  'entity_type' => 'node',
  // Создать метку для поля с использованием имени словаря.
  'label' => $vocabulary->name,
  // Установить в качестве типа контента статью.
  'bundle' => 'article',
  // Использовать текст справки словаря в качестве описания.
  'description' => $vocabulary->help,
  // Использовать виджет автозавершения таксономии.
  'widget' => array(
    'type' => 'taxonomy_autocomplete',
    'weight' => 4,
  ),
  // Определить, как элементы будут отображаться в режимах полной ноды и тизера.
  'display' => array(
    'default' => array(
      'type' => 'taxonomy_term_reference_link',
      'weight' => 10,
    ),
    'teaser' => array(
      'type' => 'taxonomy_term_reference_link',
      'weight' => 10,
    ),
  ),
);

// Присвоить полю тип контента статьи с использованием API-вызова
field_create_instance.
field_create_instance($instance);
```

Еще одно требование к улучшенному инсталляционному профилю заключается в том, что типы контента статьи и новостей должны содержать поле изображения для загрузки картинок. Сначала создадим поле, используя следующий код:

```
$field = array(
  // Определить имя поля.
  'field_name' => 'field_image',
  // Установить image в качестве типа.
  'type' => 'image',
  // Разрешить по одному изображению на ноду.
  'cardinality' => 1,
  'translatable' => TRUE,
  'locked' => FALSE,
  // Установить идентификатор файла в качестве индекса.
  'indexes' => array('fid' => array('fid')),
  'settings' => array(
    'uri_scheme' => 'public',
    'default_image' => FALSE,
  ),
);
```

```
// Определить способ хранения поля.
'storage' => array(
  'type' => 'field_sql_storage',
  'settings' => array(),
),
);
// Создать поле с использованием API-вызова field_create_field.
field_create_field($field);
```

После того как поле определено, его можно присвоить типам контента статьи и новостей:

```
$instance = array(
  // Использовать только что созданное поле изображения.
  'field_name' => 'field_image',
  // Назначить его ноду.
  'entity_type' => 'node',
  'label' => 'Image',
  // Установить в качестве типа контента статью.
  'bundle' => 'article',
  'description' => 'Upload an image to go with this article.',
  'required' => FALSE,
  // Определить настройки, связанные с изображением, которое должно быть загружено.
  'settings' => array(
    'file_directory' => 'field/image',
    'file_extensions' => 'png gif jpg jpeg',
    'max_filesize' => '',
    'max_resolution' => '',
    'min_resolution' => '',
    'alt_field' => TRUE,
    'title_field' => '',
  ),
  // Определить тип виджета, который будет использоваться.
  'widget' => array(
    'type' => 'image_image',
    'settings' => array(
      'progress_indicator' => 'throbber',
      'preview_image_style' => 'thumbnail',
    ),
    'weight' => -1,
  ),
  // Определить способ вывода изображений в представлениях полной ноды и тизера.
  'display' => array(
    'default' => array(
      'label' => 'hidden',
      'type' => 'image__large',
      'weight' => -1,
    ),
    'teaser' => array(
      'label' => 'hidden',
      'type' => 'image_link_content__medium',
      'weight' => -1,
    ),
  ),
);
```

```

// Присвоить изображение типу контента статьи,
// используя API-вызов field_create_instance.
field_create_instance($instance);

$instance = array(
  // Использовать только что созданное поле изображения.
  'field_name' => 'field_image',

  // Назначить его ноду.
  'entity_type' => 'node',
  'label' => 'Image',

  // Установить в качестве типа контента новости.
  'bundle' => 'news',
  'description' => 'Upload an image to go with this news item.',
  'required' => FALSE,

  // Определить настройки, связанные с изображением, которое должно быть загружено.
  'settings' => array(
    'file_directory' => 'field/image',
    'file_extensions' => 'png gif jpg jpeg',
    'max_filesize' => '',
    'max_resolution' => '',
    'min_resolution' => '',
    'alt_field' => TRUE,
    'title_field' => '',
  ),

  // Определить тип виджета, который будет использоваться.
  'widget' => array(
    'type' => 'image_image',
    'settings' => array(
      'progress_indicator' => 'throbber',
      'preview_image_style' => 'thumbnail',
    ),
    'weight' => -1,
  ),

  // Определить способ вывода изображений в представлениях полной ноды и тизера.
  'display' => array(
    'default' => array(
      'label' => 'hidden',
      'type' => 'image__large',
      'weight' => -1,
    ),
    'teaser' => array(
      'label' => 'hidden',
      'type' => 'image_link_content__medium',
      'weight' => -1,
    ),
  ),
);

// Связать изображение с типом контента новостей,
// используя API-вызов field_create_instance.
field_create_instance($instance);

```

Далее следует включить полномочия по умолчанию, которые должны быть связаны с системными ролями (анонимные и аутентифицируемые пользователи — систем-

ные роли по умолчанию). Сначала создадим переменную, которую можно использовать для присваивания входного формата по умолчанию анонимным и аутентифицируемым пользователям. В рассматриваемом примере для обеих ролей будет применяться фильтрованный HTML:

```
$filtered_html_permission = filter_permission_name($filtered_html_format);
```

Теперь присвоим общее разрешение на получения доступа к контенту и используем входной фильтр фильтрованного HTML для роли анонимного пользователя:

```
user_role_grant_permissions(DRUPAL_ANONYMOUS_RID,  
  array('access content', $filtered_html_permission));
```

По отношению к роли аутентифицированного пользователя мы будем щедрее. Ей мы предоставим права доступа к контенту, доступа к комментариям, размещения комментариев без утверждения и входной фильтр фильтрованного HTML:

```
user_role_grant_permissions(DRUPAL_AUTHENTICATED_RID, array('access content',  
  'access comments', 'post comments', 'post comments without approval',  
  $filtered_html_permission));
```

Следующий шаг процесса — установка роли по умолчанию для администраторов сайта и предоставление этой роли всех полномочий:

```
$admin_role = new stdClass();  
$admin_role->name = 'administrator';  
$admin_role->weight = 2;  
// Сохранить роль в базе данных Drupal.  
user_role_save($admin_role);  
// Предоставить роли администратора все полномочия.  
user_role_grant_permissions($admin_role->rid,  
  array_keys(module_invoke_all('permission')));  
// Установить эту роль в качестве роли администратора.  
variable_set('user_admin_role', $admin_role->rid);
```

После того как роль администратора создана, ее следует назначить пользователю с идентификатором пользователя (UID) 1. Для этого достаточно вставить в таблицу ролей пользователей строку, используя UID пользователя-администратора (1) и идентификатор роли администратора, созданный на предыдущем шаге:

```
db_insert('users_roles')  
  ->fields(array('uid' => 1, 'rid' => $admin_role->rid))  
  ->execute();
```

После выполнения всех описанных изменений необходимо перестроить меню, вызвав функцию `menu_rebuild`:

```
menu_rebuild();
```

Последний шаг процесса — установка и включение темы `seven` в качестве административной темы и закрытие функции инсталляции:

```
db_update('system')  
  ->fields(array('status' => 1))  
  ->condition('type', 'theme')  
  ->condition('name', 'seven')  
  ->execute();  
variable_set('admin_theme', 'seven');  
variable_set('node_admin_theme', '1');  
// Закрыть функцию инсталляции.  
}
```

Теперь улучшенный инсталляционный профиль завершен и готов к использованию для установки Drupal. Расширение профиля инсталляции можно было бы продолжить, добавляя в него дополнительные блоки, создавая дополнительные типы контента и поля, назначая этим полям типы контента, создавая словари таксономии и пользовательские роли.

Использование хуков `hook_install_tasks` и `hook_install_tasks_alter`

В ряде ситуаций желательно увидеть список задач, которые будут выполняться во время инсталляции или модифицировать задачи, определенные в профиле. Для получения снабженного ключами массива задач, которые профиль выполнит во время заключительного этапа установки, можно применить хук `hook_install`. Каждый ключ представляет имя функции (обычно функции, определенной данным профилем, хотя это и не обязательно), вызываемой при выполнении данной задачи. Для получения дополнительной информации о хуке `hook_install` обратитесь по адресу http://api.drupal.org/api/function/hook_install_tasks/7.

Для изменения задач, которые выполняются во время инсталляции, можно использовать хук `hook_install_tasks_alter`. Сигнатура функции этого хука — `hook_install_tasks_alter(&$tasks, $install_state)`; параметры описаны ниже.

- `$tasks` — массив всех доступных задач установки, включая предоставленные ядром Drupal. Этот массив можно модифицировать, чтобы изменить или заменить любую часть процесса установки Drupal, выполняемого после того, как инсталляционный профиль выбран.
- `$install_state` — массив информации о текущем состоянии установки.

Модуль демонстрационного профиля Demo Profile (http://drupal.org/project/demo_profile) использует хук `hook_install_tasks_alter` для удаления некоторых задач установки, определенных в инсталляционном профиле. Это связано с тем, что модуль устанавливает резервную копию существующего сайта (<http://drupal.org/project/demo>) в качестве исходного варианта нового сайта. Поскольку резервная копия базы данных включает многие из таблиц, необходимых для приведения сайта в рабочее состояние, нежелательно, чтобы процесс установки стер эти таблицы. Как видно в следующем примере кода, модуль удаляет такие задачи, как инсталляция ядра (`install_system_module`), инсталляция пользователей (`install_profile_modules`), инсталляция локалей (`install_import_locales`) и т.д.

```
function demo_profile_install_tasks_alter(&$tasks, &$install_state) {
  // Сохранить начальную загрузку и установите окончательные варианты задач —
  // мы снова их используем позже в процессе инсталляции.
  $install_bootstrap_full = (array) $tasks['install_bootstrap_full'];
  $install_finished = (array) $tasks['install_finished'];

  // Удалить задачи из списка и процесса выполнения.
  // Мы не можем реализовать hook_install_tasks(), поскольку процесс
  // установки требуется перехватить еще до его начала (за исключением
  // настроек базы данных).
  unset(
    $tasks['install_system_module'],
    $tasks['install_bootstrap_full'],
    $tasks['install_profile_modules'],
    $tasks['install_import_locales'],
    $tasks['install_configure_form'],
```

```

$tasks['install_import_locales_remaining'],
$tasks['install_finished']
);

// Добавить задачи демонстрационного профиля сайта.
// @todo Выделить настройку пути подкачки в отдельный шаг;
// сохранить значение в $install_state.
$tasks['demo_profile_form'] = array(
  'display_name' => st('Choose snapshot'),
  'type' => 'form',
  'run' => INSTALL_TASK_RUN_IF_NOT_COMPLETED,
);
// Выполнить полную начальную загрузку и вывести на экран финальное сообщение.
$tasks['install_bootstrap_full'] = $install_bootstrap_full;
$tasks['install_finished'] = $install_finished;
}

```

Резюме

Если одни и те же типы сайтов приходится строить снова и снова, рекомендуется подумать об использовании инсталляционных профилей Drupal в качестве средств ускорения процесса создания сайта. Инсталляционные профили автоматизируют многие из задач, связанных с установкой и конфигурированием сайта Drupal. Устанавливая и конфигурируя одну и ту же структуру сайта раз за разом, легко пропустить тот или иной шаг. Инсталляционные профили устраняют эту проблему, автоматизируя процесс.

В этой главе были описаны файлы, связанные с созданием нового инсталляционного профиля, структура и содержимое каждого из этих файлов, а также особенности параметров конфигурации, связанных с созданием и включением основных функциональных средств, таких как блоки, типы контента, поля, таксономия и роли пользователей.

ГЛАВА 25

Тестирование

Одним из замечательных новых средств, включенных в ядро Drupal 7, является платформа тестирования. Теперь ядро содержит интегрированный инструмент тестирования, который позволяет определять и автоматизировать тестирование своего сайта Drupal. Платформа тестирования Drupal 7 позволяет автоматически выполнять сотни, а то и тысячи тестов, которые обеспечивают уверенность в том, что внесенные в сайт изменения не приведут к нарушению его работы.

В этой главе будет показано, как активизировать платформу тестирования и определять, выполнять, отслеживать и просматривать результаты тестов.

Настройка тестовой среды

Первый шаг в настройке тестовой среды — включение модуля `Testing` в ядре Drupal 7. Перейдите к странице `Modules` (Модули) и выполните прокрутку вниз до модуля `Testing`. Отметьте флажок рядом с модулем и щелкните на кнопке `Save configuration` (Сохранить конфигурацию). Теперь инструменты тестирования в вашем распоряжении и готовы к использованию.

Прежде чем приступить к созданию собственного набора тестов, рассмотрим тесты, которые поставляются с Drupal 7. Одна из основных причин, по которым Drupal 7 является одной из самых устойчивых версий Drupal, связана с применением платформы тестирования для обеспечения правильной работы ядра Drupal в целом. Тестирование ядра Drupal старым способом заняло бы месяцы и требовало бы “замораживания” кода на долгие месяцы, предшествующие запуску, чтобы хватило времени для тщательного тестирования изменений и улучшений ядра. Использование платформы тестирования позволило команде разработчиков определить набор тестов, которые, несомненно, доказывали бы, что ядро Drupal делает то, чего от него ожидают, и делает это снова и снова после внесения изменений в код, который составляет ядро Drupal.

Рассмотрим тесты, которые поставляются с ядром Drupal 7, перейдя на страницу `Configuration⇒Testing` (Конфигурация⇒Тестирование). Эта страница (рис. 25.1) содержит длинный список модулей ядра Drupal 7 и фактически представляет собой список модулей и подсистем ядра Drupal 7, например, `AJAX` или `Batch API`. Одной из задач группы разработки ядра Drupal 7 было определение для каждого модуля ядра набора тестов, которые можно было бы выполнить для подтверждения того, что каждый модуль дает ожидаемые от него результаты.

Чтобы удостовериться, насколько легко выполнить комплект тестов в среде Drupal 7, в качестве примера запустим тесты, связанные с модулем `Blog`. Просто отметьте флажок рядом с модулем `Blog`, выполните прокрутку в конец страницы и щелкните на кнопке `Run tests` (Выполнить тесты). Немедленно после щелчка на этой кнопке среда Drupal незаметно выполняет ряд действий по подготовке к тесту, как показано на рис. 25.2.

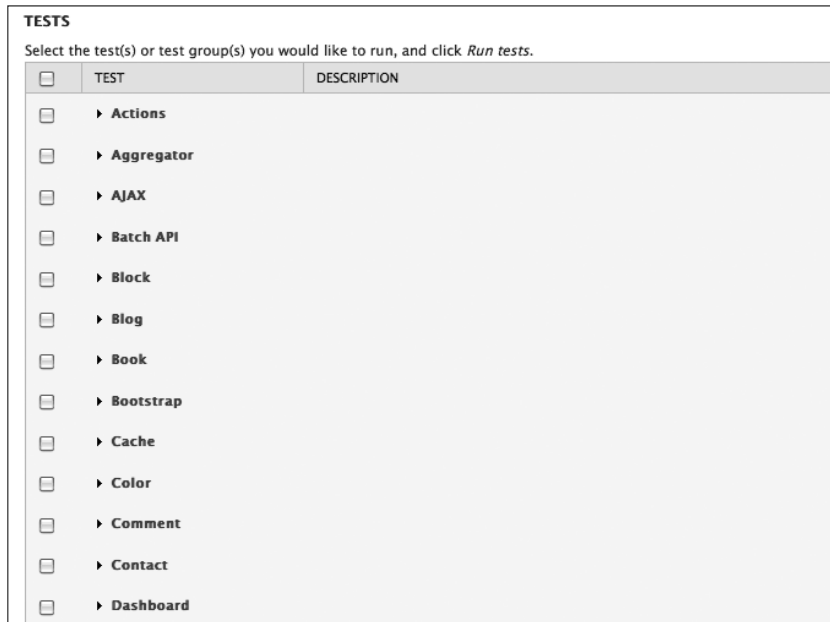


Рис. 25.1. Готовые тесты, поставляемые в составе ядра Drupal 7

Сначала она создает полную инсталляцию Drupal. Этот подход ведет к созданию новой среды при каждом выполнении комплекта тестов, обеспечивая идентичность начальных условий при каждом выполнении теста. Затем тестовая платформа использует PHP cURL для выполнения предопределенной серии тестов, как если бы пользователь прогонял эти тесты в браузере вручную, и записывает и выводит результаты тестов на экран. Важно отметить, что при каждом выполнении серии тестов платформа тестирования запускается с чистой инсталляцией Drupal, а по завершении теста очищает использованные для выполнения файлы и таблицы. В процессе настройки следует определить, какие модули должны быть установлены и активированы, какие учетные записи пользователя должны быть созданы и какие типы контента должны присутствовать. Фактически большинство шагов, которые нужно было бы выполнить для настройки сайта, потребуется выполнить и при создании сценария процесса настройки теста, что вскоре будет подробно описано. Преимущество этого подхода в том, что каждый раз запуск выполняется с “чистого листа”, гарантируя полную повторяемость одних и тех же тестов, поскольку они запускаются из той же самой базовой среды, а не из среды, которая продолжает изменяться со временем вследствие других действий, связанных с тестированием, к тому же сайт разработки или тестирования не засоряется данными тестирования.

В конце тестового цикла Drupal демонтирует тестовую среду и выводит на экран итоговую страницу, которая сообщает количество выполненных тестов, число невыполненных тестовых условий, количество полученных сообщений об исключениях и число сообщений отладки, которые были получены во время процесса тестирования. Результаты выполнения тестов модуля Blog приведены на рис. 25.3.

Как видно из результатов тестов, 241 тест был выполнен за 1 минуту и 44 секунды, и все тесты прошли успешно. Попытка выполнения этих тестов вручную без какой-либо автоматизации тестирования заняла бы не один час и была бы сопряжена с риском потенциальных присутствующих человеку ошибок во время прогона тестов.

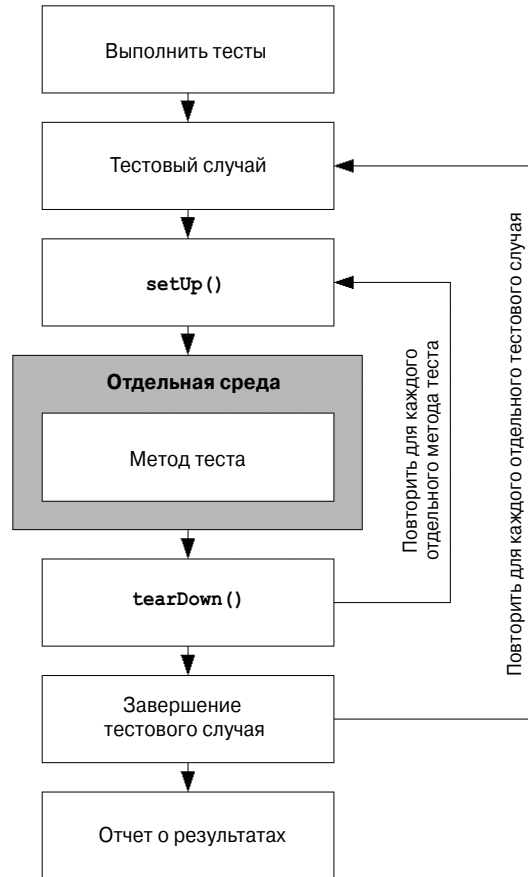


Рис. 25.2. Процесс тестирования в Drupal 7

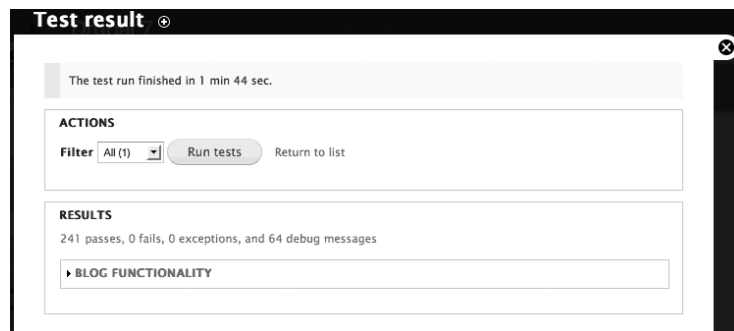


Рис. 25.3. Результаты выполнения тестов модуля Blog

Чтобы просмотреть подробный отчет о результатах тестов, щелкните на ссылке Blog Functionality (Функциональность модуля Blog), которая выводит список выполненных тестов и подробные результаты каждого теста, которые были зарегистрированы во время его выполнения (рис. 25.4).

RESULTS					
241 passes, 0 fails, 0 exceptions, and 64 debug messages					
▼ BLOG FUNCTIONALITY					
Create, view, edit, delete, and change blog entries and verify its consistency in the database.					
241 passes, 0 fails, 0 exceptions, and 64 debug messages					
MESSAGE	GROUP	FILENAME	LINE	FUNCTION	STATUS
Created role of name: cMEWJrzP, id: 4	Role	blog.test	23	BlogTestCase->setUp()	✓
Created permissions: administer blocks	Role	blog.test	23	BlogTestCase->setUp()	✓
User created with name eCSLNyaS and pass VV3NMrmdeF	User login	blog.test	23	BlogTestCase->setUp()	✓
Created role of name: sL1KxuOj, id: 5	Role	blog.test	24	BlogTestCase->setUp()	✓
Created permissions: create blog content, edit own blog content, delete own blog content	Role	blog.test	24	BlogTestCase->setUp()	✓
User created with name qId4a11i and pass 5pHgV94r4F	User login	blog.test	24	BlogTestCase->setUp()	✓
Created role of name: pzenrCHH, id: 6	Role	blog.test	25	BlogTestCase->setUp()	✓

Рис. 25.4. Список выполненных тестовых условий и их состояния

Скорее всего, список тестовых условий будет содержать ссылки Verbose messages (Подробные сообщения), как показано на рис. 25.5. Как правило, эти сообщения включают снимки экрана результатов тестового случая, представляя материальное доказательство того, что сценарий тестирования сделал то, чего от него ожидали, что полезно при отладке тестов.

MESSAGE	GROUP	FILENAME	LINE	FUNCTION	STATUS
Verbose message	Debug	blog.test	37	BlogTestCase->testUnprivilegedUser()	⚠
GET http://localhost/user/2 returned 200 (5.33 KB).	Browser	blog.test	37	BlogTestCase->testUnprivilegedUser()	✓
Valid HTML found on "http://localhost/user/2"	Browser	blog.test	37	BlogTestCase->testUnprivilegedUser()	✓
Verbose message	Debug	blog.test	37	BlogTestCase->testUnprivilegedUser()	⚠
User eCSLNyaS successfully logged in.	User login	blog.test	37	BlogTestCase->testUnprivilegedUser()	✓
GET http://localhost/blog/2 returned 200 (6.24 KB).	Browser	blog.test	39	BlogTestCase->testUnprivilegedUser()	✓
Valid HTML found on "http://localhost/blog/2"	Browser	blog.test	39	BlogTestCase->testUnprivilegedUser()	✓
Verbose message	Debug	blog.test	39	BlogTestCase->testUnprivilegedUser()	⚠

Рис. 25.5. Результаты тестов со ссылками Verbose messages предоставляют дополнительные подробности, такие как снимки экрана

Щелчок на ссылке *Verbose message* в строке 39 отображает сохраненную/статическую версию XHTML-вывода, которая сохранена на момент выполнения данного тестового условия (рис. 25.6), предоставляя несомненное доказательство, что сайт обеспечил ожидаемые результаты, определенные в тестовом условии.



Рис. 25.6. Снимок экрана, полученный сценарием тестирования и показывающий фактические результаты

Платформа тестирования обеспечивает также возможность вывода на экран значений, связанных с переменными и объектами, такими как объект ноды, обеспечивая дополнительную помощь в устранении причин неудачи теста. Если подтверждение сравнить с пудингом, то платформа тестирования Drupal — двухслойный шоколадный пудинг, богатый вкусовыми оттенками и вызывающий восхищение у гурманов.

Способ определения тестов

Как правило, тесты связаны с модулем, и как таковые определяются на уровне модуля. В предыдущем случае теста модуля *Blog* разработчики модуля *Blog* создали в каталоге этого модуля новый файл *blog.test*. Содержимое модуля *blog.test* отражает параметры настройки среды разработчиком и индивидуальные тестовые условия, которые будут выполняться при прогоне комплекта тестов.

Прежде всего, следует отметить, что тестирование является объектно-ориентированным. По существу тест — это объект, который создается из класса, основанного на классах *DrupalWebTestCase* или *DrupalUnitTestCase*. Принимая объектно-ориентированный подход, наш тестовый класс наследует всю функциональность, определенную в базовом классе, позволяя разработчику сосредоточиться на тех тестах, которые желательно выполнить, а не на кодировании сценария для обработки таких задач, как загрузка страницы, определение способов ввода текста в форму и т.д.

Следующий шаг, который нужно выполнить — определение переменных экземпляров, которые будут использоваться в ходе всего процесса тестирования. В случае тестов модуля *Blog* разработчик определил три переменные, каждая из которых представляет определенного пользователя с определенными назначенными ему полномочиями.


```
<?php

class BlogTestCase extends DrupalWebTestCase {
    protected $big_user;
    protected $own_user;
    protected $any_user;
```

Далее необходимо определить с помощью функции `getInfo` имя, описание и группу, связанные с данным набором тестов. Именно эта информация отображается на странице `Configuration⇒Testing` (Конфигурация⇒Тестирование). В поле группирования этот тест расположен в группе `Blog`. Чтобы увидеть имя и описание теста, щелкните на стрелке слева от элемента `Blog` на странице `Tests` (Тесты). Этот подход позволяет определить логические наборы тестов и связать эти тесты с определенным модулем.

```
public static function getInfo() {
    return array(
        'name' => 'Blog functionality',
        'description' => 'Create, view, edit, delete, and change blog entries and
        verify its consistency in the database.',
        'group' => 'Blog',
    );
}
```

Следующий раздел, определенный разработчиком, представляет собой процесс настройки. Когда процесс тестирования начинается, в первую очередь тестовая платформа создает новый базовый экземпляр `Drupal`. Если наши тесты требуют модулей, учетных записей пользователя, типов контента, файлов в каталоге или что-то еще помимо базовой инсталляции ядра `Drupal 7`, для выполнения этих задач следует использовать функцию `setUp()`. В случае тестов модуля `Blog` единственное, что требуется выполнить — сначала активизировать модуль `Blog`, что выполняется вызовом функции `parent::setUp('blog')`. При наличии дополнительных модулей, которые должны были быть активированы, каждый модуль нужно было бы добавить в качестве дополнительного параметра, такого как `parent::setUp('blog', 'ctools', 'panels', 'date')`.

Следующий шаг создает три учетных записи пользователя, которые мы хотим использовать, назначая каждой из них определенные полномочия. Мы создаем нового пользователя, выполняя метод `drupalCreateUser`, связанный с текущим объектом тестирования, ссылка на который выглядит как `$this`. Чтобы назначить полномочия, достаточно передать массив полномочий, которые требуется выдать каждой учетной записи пользователя.

```
/**
 * Активирование модулей и создание пользователей
 * с определенными полномочиями.
 */
function setUp() {
    parent::setUp('blog');
    // Создание пользователей.
    $this->big_user = $this->drupalCreateUser(array('administer blocks'));
    $this->own_user = $this->drupalCreateUser(array('create blog content',
        'edit own blog content', 'delete own blog content'));
    $this->any_user = $this->drupalCreateUser(array('create blog content',
        'edit any blog content', 'delete any blog content',
        'access administration pages'));
}
```

Следующая функция действует в качестве основного контроллера выполнения тестов.

На заметку! Любая функция, запускаемая с тестом, будет автоматически обнаружена и выполнена.

В этой функции сценарий тестирования регистрирует на сайте пользователя-администратора `big_user`, после чего активирует блок последней публикации в блоге и назначает его области `sidebar_second`. Следующий шаг конфигурирует блок последних публикаций в блоге, устанавливая количество сообщений, которые будут отображаться в блоке, равным пяти. Следующий шаг выполняет функцию `doBasicTests` (она определена позже), используя учетные записи `any_user` и `own_user`, а затем выполняет другие тестовые функции, определенные где-либо в данном сценарии тестирования.

```
/**
 * Вход пользователей, создание нод блога и тестирование функциональности
 * блога с помощью интерфейсов администратора и пользователя.
 */

function testBlog() {

    // Вход пользователя-администратора.
    $this->drupalLogin($this->big_user);

    // Активация последнего блока блога.
    $edit = array();
    $edit['blog_recent[region]'] = 'sidebar_second';
    $this->drupalPost('admin/structure/block', $edit, t('Save blocks'));
    $this->assertResponse(200);

    // Проверка возможности изменения количества последних
    // сообщений блога в блоке.
    $edit = array();
    $edit['blog_block_count'] = 5;
    $this->drupalPost('admin/structure/block/manage/blog/recent/configure',
        $edit, t('Save block'));
    $this->assertEqual(variable_get('blog_block_count', 10), 5,
        t('Number of recent blog posts changed.));

    // Выполнение базовых тестов для каждого пользователя.
    $this->doBasicTests($this->any_user, TRUE);
    $this->doBasicTests($this->own_user, FALSE);

    // Создание еще одной ноды блога для любого пользователя блога.
    $node = $this->drupalCreateNode(array('type' => 'blog',
        'uid' => $this->any_user->uid));

    // Проверка, что только у пользователя-владельца блога
    // имеется доступ к ноде просмотра блога.
    $this->verifyBlogs($this->any_user, $node, FALSE, 403);

    // Создание еще одной ноды блога для пользователя-владельца блога.
    $node = $this->drupalCreateNode(array('type' => 'blog',
        'uid' => $this->own_user->uid));

    // Вход любого пользователя блога.
    $this->drupalLogin($this->any_user);

    // Проверка, что любой пользователь блога обладает доступом
    // ко всем нодам блога.
    $this->verifyBlogs($this->own_user, $node, TRUE);
}
```

Следующий набор функций выполняет определенные функциональные тесты. Функция `testUnprivilegedUser` удостоверяется, что у кого-либо, не имеющего соответствующих прав, отсутствует возможность отправлять сообщения в блог на данном веб-сайте. Процесс тестирования этой функциональной возможности начинается с попытки создания ноды посредством метода `drupalCreateNode`, который принимает массив параметров, используемых при попытке создания ноды. Следующий шаг — регистрация в сайте с помощью метода `drupalLogin` и учетной записи `big_user`, созданной в функции `setup`. После того, как пользователь зарегистрирован, следующий шаг заключается в переходе к странице блога пользователя `big_user` с помощью метода `drupalGet`. Следующий шаг проверяет код HTTP-ответа, возвращенный в результате попытки перехода к странице блога этого пользователя. Для тестирования успешности перехода пользователя к странице своего блога разработчик применял метод `assertResponse` для проверки кода HTTP-ответа. Если кодом ответа является 200, это свидетельствует об успешном прохождении теста и регистрации его как такового. Разработчик продолжает тест, проверяя настройку имени пользователя и слова "blog" в качестве заголовка страницы, используя метод `assertTitle`; если это так, регистрируется успешное выполнение теста. Заключительный тест заключается в проверке отображения текста "You are not allowed to post a new blog entry" ("Вам запрещена отправка новой записи блога") где-либо на странице. Для этого разработчик использует функцию `assertText`, которая сканирует страницу, отыскивая точное совпадение со строкой, переданной в качестве первого параметра. Наличие соответствия свидетельствует об успешном выполнении теста, и результат регистрируется с использованием второго параметра, переданного в метод `assertText`.

```
/**
 * Подтверждение того, что сообщение "You are not allowed to post a new blog
 * entry" отображается, если пользователь, отправивший записи в блог,
 * нарушает данное разрешение и обращается к странице блога.
 */
function testUnprivilegedUser() {
  // Создание ноды блога для пользователя без каких-либо полномочий в блоге.
  $this->drupalCreateNode(array('type' => 'blog',
    'uid' => $this->big_user->uid));

  $this->drupalLogin($this->big_user);

  $this->drupalGet('blog/' . $this->big_user->uid);
  $this->assertResponse(200);
  $this->assertTitle(t("@name's blog",
    array('@name' => format_username($this->big_user)))
    . ' | Drupal', t('Blog title was displayed'));
  $this->assertText(t('You are not allowed to post a new blog entry.'),
    t('No new entries can be posted without the right permission'));
}
```

Следующий тест проверяет, может ли пользователь просматривать страницу блога другого пользователя, когда у этого другого пользователя нет никаких сообщений в блоге. Процесс подобен предыдущему тесту. Разработчик входит на сайт, используя метод `drupalLogin` и передавая `big_user` в качестве регистрирующегося пользователя. Следующий шаг заключается в переходе к странице блога пользователя `own_user` посредством метода `drupalGet`. С помощью метода `assertResponse` разработчик проверяет успешность загрузки страницы, а затем с применением метода `assertTitle` — правильности отображения ее заголовка. Текст должным образом выводится на экран для страницы пользователя блога, когда у данного пользователя нет никаких записей в блоге (для этой проверки используется метод `assertText`).

```

/**
 * Просмотр блога пользователя, не имеющего записей блога,
 * другим пользователем.
 */
function testBlogPageNoEntries() {
    $this->drupalLogin($this->big_user);
    $this->drupalGet('blog/' . $this->own_user->uid);
    $this->assertResponse(200);
    $this->assertTitle(t("@name's blog",
        array('@name' => format_username($this->own_user)))
        . ' | Drupal', t('Blog title was displayed'));
    $this->assertText(t('@author has not created any blog entries.',
        array('@author' => format_username($this->own_user))),
        t('Users blog displayed with no entries'));
}

```

Функция `doBasicTests` выполняет другие тестовые функции, определенные в данном сценарии тестирования. Функция регистрирует пользователя и выполняет тесты, которые проверяют блоги, создает ноды и проверяет ссылки блога.

```

/**
 * Выполнение базовых тестов для указанного пользователя.
 *
 * @param object $user
 *   Зарегистрированный пользователь.
 * @param boolean $admin
 *   Пользователь имеет полномочия 'access administration pages'.
 */
private function doBasicTests($user, $admin) {
    // Вход пользователя.
    $this->drupalLogin($user);

    // Создание ноды блога.
    $node = $this->drupalCreateNode(array('type' => 'blog'));

    // Проверка, что пользователь обладает доступом ко всем нодам блога.
    $this->verifyBlogs($user, $node, $admin);

    // Создание еще одной ноды, чтобы протестировать страницу блога
    // более чем с одной нодой.
    $this->drupalCreateNode(array('type' => 'blog', 'uid' => $user->uid));

    // Проверка отображения ссылок блога.
    $this->verifyBlogLinks($user);
}

```

Следующие функции продолжают использовать только определенные подходы для выполнения различных тестов и проверки и записи результатов.

```

/**
 * Проверка, что зарегистрированный пользователь обладает
 * требуемым доступом к различным нодам блога.
 *
 * @param object $node_user
 *   Пользователь, создавший ноду.
 * @param object $node
 *   объект ноды.
 * @param boolean $admin
 *   Пользователь имеет полномочия 'access administration pages'.

```

```

* @param integer $response
*   Код HTTP-ответа.
*/
private function verifyBlogs($node_user, $node, $admin, $response = 200)
{
    $response2 = ($admin) ? 200 : 403;

    // Просмотр ноды справки блога.
    $this->drupalGet('admin/help/blog');
    $this->assertResponse($response2);
    if ($response2 == 200) {
        $this->assertTitle(t('Blog | Drupal'), t('Blog help node was displayed'));
        $this->assertText(t('Blog'), t('Blog help node was displayed'));
    }

    // Проверка отображения блока блога.
    $this->drupalGet('');
    $this->assertResponse(200);
    $this->assertText(t('Recent blog posts'), t('Blog block was displayed'));

    // Просмотр ноды блога.
    $this->drupalGet('node/' . $node->nid);
    $this->assertResponse(200);
    $this->assertTitle($node->title . ' | Drupal', t('Blog node was displayed'));
    $breadcrumb = array(
        1(t('Home'), NULL),
        1(t('Blogs'), 'blog'),
        1(t('!name's blog', array('!name' => format_username($node_user)),
            'blog/' . $node_user->uid),
    );

    $this->assertRaw(theme('breadcrumb', array('breadcrumb' => $breadcrumb)),
        t('Breadcrumbs were displayed'));

    // Просмотр ноды редактирования блога.
    $this->drupalGet('node/' . $node->nid . '/edit');
    $this->assertResponse($response);
    if ($response == 200) {
        $this->assertTitle('Edit Blog entry ' . $node->title . ' | Drupal',
            t('Blog edit node was displayed'));
    }

    if ($response == 200) {
        // Редактирование ноды блога.
        $edit = array();
        $langcode = LANGUAGE_NONE;
        $edit["title"] = 'node/' . $node->nid;
        $edit["body"][$langcode][0][value] = $this->randomName(256);
        $this->drupalPost('node/' . $node->nid . '/edit', $edit, t('Save'));
        $this->assertRaw(t('Blog entry %title has been updated.',
            array('%title' => $edit["title"])), t('Blog node was edited'));

        // Удаление ноды блога.
        $this->drupalPost('node/' . $node->nid . '/delete', array(), t('Delete'));
        $this->assertResponse($response);
        $this->assertRaw(t('Blog entry %title has been deleted.',
            array('%title' => $edit["title"])), t('Blog node was deleted'));
    }
}

```

```

/**
 * Проверка, что ссылки блога отображаются зарегистрированному пользователю.
 *
 * @param object $user
 *   Зарегистрированный пользователь.
 */
private function verifyBlogLinks($user) {
    // Подтверждение, что ссылка записей блога существует на странице пользователя.
    $this->drupalGet('user/' . $user->uid);
    $this->assertResponse(200);
    $this->assertText(t('View recent blog entries'),
        t('View recent blog entries link was displayed'));

    // Подтверждение, что ссылка последних записей блога поступает
    // на страницу блога пользователя.
    $this->clickLink('View recent blog entries');
    $this->assertTitle(t("@name's blog | Drupal",
        array('@name' => format_username($user))),
        t('View recent blog entries link target was correct'));

    // Подтверждение отображения страницы блога.
    $this->drupalGet('blog');
    $this->assertResponse(200);
    $this->assertTitle('Blogs | Drupal', t('Blog page was displayed'));
    $this->assertText(t('Home'), t('Breadcrumbs were displayed'));
    $this->assertLink(t('Create new blog entry'));

    // Подтверждение отображения страницы блога для отдельного пользователя.
    $this->drupalGet('blog/' . $user->uid);
    $this->assertTitle(t("@name's blog | Drupal",
        array('@name' => format_username($user))),
        t('User blog node was displayed'));

    // Подтверждение отображения источника блога.
    $this->drupalGet('blog/feed');
    $this->assertTitle(t('Drupal blogs'), t('Blog feed was displayed'));

    // Подтверждение отображения источника блога для отдельного пользователя.
    $this->drupalGet('blog/' . $user->uid . '/feed');
    $this->assertTitle(t("@name's blog",
        array('@name' => format_username($user))),
        t('User blog feed was displayed'));
}
}

```

Тестовые функции

В сценарии тестирования блога применялось несколько тестовых функций, входящих в состав платформы тестирования. Приступая к документированию функциональности, которую требуется протестировать на своем сайте, нужно будет определить “способ” выполнения соответствующих тестов. “Способ” будет увязывать воедино одну или более функций, которые имитируют действия конечного пользователя, выполняющего функции на сайте. Соответствующие утверждения подтверждают, что выполняемые посредством этих функций тесты обеспечивают ожидаемые результаты. Функции, которые можно использовать, описаны в табл. 25.1, а утверждения, используемые для проверки результатов — в табл. 25.2.

Таблица 25.1. Функции тестирования

Функция	Описание
<code>\$this->drupalGet(\$path, \$options=array())</code>	Эта функция выполняет запрос <code>get</code> применительно к URL на сайте. <code>\$path</code> указывает страницу, которая будет посещена, а <code>\$options</code> содержит дополнительные данные, которые можно передать URL, чтобы определить URL для посещения. Контент будет загружен и сохранен в <code>\$this->_content</code> , откуда он может быть получен с помощью функции <code>\$this->drupalGetContent()</code> .
<code>\$this->drupalPost(\$path, \$edit, \$submit, \$reporting=TRUE)</code>	Эта функция выполняет запрос <code>post</code> применительно к URL на сайте. <code>\$path</code> указывает страницу, содержащую форму, которая будет заполнена данными, описанными в параметре <code>\$edit</code> . Затем будет выполнен щелчок на кнопке, указанной параметром <code>\$submit</code> . Данные параметра <code>\$edit</code> должны быть массивом, где каждый индекс — значение атрибута <code>name</code> элемента HTML-формы.
<code>\$this->clickLink(\$label, \$index=0)</code>	Эта функция следует за ссылкой на текущей странице. Параметр <code>\$label</code> должен содержать текст, связанный со ссылкой. При наличии на странице нескольких ссылок с одинаковым текстом для указания ссылки, на которой должен быть выполнен щелчок, используйте параметр <code>\$index</code> , который подсчитывает на странице количество ссылок с одинаковым текстом, просматривая источник страницы и подсчитывая ссылки, начиная сверху.
<code>\$this->drupalCreateUser(\$permissions = NULL)</code>	Эта функция создает нового пользователя и назначает полномочия, перечисленные в параметре <code>\$permissions</code> (например, <code>array('access comments', 'access content', 'post comments')</code>). Функция возвращает полностью заполненный пользовательский объект Drupal с дополнительным значением <code>pass_raw</code> , которое содержит нехешированный пароль. Функция также создает пользовательскую роль с указанными полномочиями и назначает эту роль учетной записи пользователя.
<code>\$this->drupalLogin(\$user = NULL)</code>	Эта функция регистрирует пользователя на сайте, используя виртуальный браузер, созданный во время процесса запуска. Параметр <code>\$user</code> — стандартный объект пользователя Drupal. Если объект пользователя не был передан функции, функция сама создаст нового пользователя.
<code>\$this->drupalLogout()</code>	Эта функция регистрирует пользователя в виртуальном браузере. Она автоматически вызывается функцией <code>\$this->drupalLogin</code> , гарантируя регистрацию только одного пользователя в данный момент времени. Учет этого нюанса чрезвычайно важен при написании сценариев — только один пользователь может быть зарегистрированным в любой данный момент времени.
<code>\$this->drupalCreateRole(\$permissions = NULL)</code>	Эта функция создает роль Drupal, используя полномочия, переданные в параметре <code>\$permissions</code> . Она возвращает значение, которое является идентификатором роли или значением <code>FALSE</code> в случае неудачи. В большинстве случаев вызов этой функции не требуется, поскольку функция <code>\$this->drupalCreateUser</code> автоматически создает новую роль на основе полномочий, определенных в ее параметре <code>\$permissions</code> .

Функция	Описание
<code>\$this->randomString(\$number = 8)</code>	Эта функция возвращает строку длины, определенной параметром <code>\$number</code> , где каждый символ лежит в пределах ASCII-кодов 32–126. Эту функцию можно использовать для создания заголовков нод, тел нод и т.д.
<code>\$this->randomName(\$number = 8)</code>	Эта функция возвращает строку длины, определенной параметром <code>\$number</code> , где каждый символ лежит в пределах ASCII-кодов 32–126. Эту функцию можно использовать, для создания заголовков нод, тел нод и т.д.
<code>\$this->drupalCreateContentType(\$settings)</code>	Эта функция создает новый нестандартный тип контента на основе настроек, определенных в параметре <code>\$settings</code> . Значения по умолчанию для нод устанавливаются автоматически. Эти настройки можно переопределить посредством параметра <code>\$settings</code> . Примером <code>\$settings</code> мог бы служить: <pre>\$settings = array('type' => 'event', 'title' => 'Event Title', 'body_label' => 'Event Description',);</pre>
<code>\$this->drupalCreateNode(\$settings)</code>	Эта функция создает новую ноду, используя значения по умолчанию. Эти настройки можно переопределить или дополнить новыми данными посредством параметра <code>\$settings</code> . Примером <code>\$settings</code> мог бы служить: <pre>\$settings = array('type' => 'event', 'event_date' => '2012-12-21 00:00:00',);</pre> где в качестве типа контента устанавливается событие, а не тип контента по умолчанию для сайта, и создается новое поле <code>event_date</code> с присваиванием ему значения. Значения по умолчанию, связанные с созданием ноды, следующие: <pre>\$defaults = array('body' => \$this->randomName(32), 'title' => \$this->randomName(8), 'comment' => 2, 'changed' => time(), 'format' => FILTER_FORMAT_DEFAULT, 'moderate' => 0, 'promote' => 0, 'revision' => 1, 'log' => '', 'status' => 1, 'sticky' => 0, 'type' => 'page', 'revisions' => NULL, 'taxonomy' => NULL,);</pre>
<code>\$this->cronRun()</code>	Эта функция запускает <code>cron</code> .

Функция	Описание
<code>\$this->drupalGetNodeByTitle(\$title)</code>	Эта функция извлекает ноду по заголовку, определенному в параметре <code>\$title</code> .
<code>\$this->drupalGetTestFiles(\$type, \$size = NULL)</code>	Эта функция возвращает список файлов, которые соответствуют типам, определенным в параметре <code>\$type</code> (например, <code>binary</code> , <code>html</code> , <code>image</code> , <code>javascript</code> , <code>php</code> , <code>sql</code> , <code>text</code>), и размер файла в байтах как определено в параметре <code>\$size</code> . Эта функция проверяет заданный по умолчанию общедоступный каталог <code>public</code> на предмет существования файлов.
<code>\$this->drupalCompareFiles(\$file1, \$file2)</code>	Эта функция выполняет сравнение файлов и возвращает различия между этими двумя файлами.
<code>\$this->checkPermissions(array \$permissions, \$reset = FALSE)</code>	Эти функции проверяет, назначен ли зарегистрированному пользователю набор полномочий, определенный в параметре <code>\$permissions</code> .
<code>\$this->refreshVariables()</code>	Эта функция сбрасывает определенные переменные в их исходное состояние.
<code>\$this->drupalHead(\$path, array \$options = array(), array \$headers = array())</code>	Эта функция возвращает только заголовки пути Drupal или абсолютного пути. Параметр <code>\$path</code> определяет URL, который должен быть загружен во внутренний браузер, параметр <code>\$options</code> определяет опции, которые будут переданы URL, а параметр <code>\$headers</code> содержит дополнительные заголовки HTTP-запроса, сформатированные как <i>имя: значение</i> .
<code>\$this->xpath(\$xpath)</code>	Эта функция выполняет поиск XPath в контенте внутреннего браузера, как определено в параметре <code>\$xpath</code> .
<code>\$this->getAllOptions(SimpleXMLElement \$element)</code>	Эта функция возвращает все элементы опций, включая вложенные опции, в выборке. Параметр <code>\$element</code> определяет элемент, для которого нужно получить опции.
<code>\$this->drupalGetMails(\$filter = array())</code>	Эта функция возвращает массив, который содержит все сообщения электронной почты, отправленные во время выполнения теста. Предназначенные для возврата сообщения электронной почты можно фильтровать, определяя парные фильтры <i>ключ/значение</i> .
<code>\$this->getSelectedItem(SimpleXMLElement \$element)</code>	Эта функция возвращает значение из элемента, определенного в параметре <code>\$element</code> .

Тестовые утверждения

Утверждения — это функции, которые проверяют истинность определенных условий, например, что поле на текущей странице содержит определенное значение. Утверждения можно считать проверкой того, что действия, выполняемые предшествующими функциями, обеспечили результаты, которые от них ожидалось. Каждое утверждение оправдывается либо не оправдывается, в зависимости от того, истинно или ложно исследуемое условие. Функции утверждения, которые можно использовать для проверки результатов тестов, описаны в табл. 25.2.

Таблица 25.2. Тестовые утверждения

Утверждение	Описание
<code>\$this->assertTrue(\$result, \$message = FALSE, \$group = 'Other')</code>	Эта функция утверждает, что в результате переменная <code>\$result</code> получает истинное значение. <code>\$age = is_integer(123); \$this->assertTrue(\$is_number, t('Make sure that the person\'s age is an integer value'));</code>
<code>\$this->assertFalse(\$result, \$message = '%', \$group = 'Other')</code>	Эта функция утверждает, что в результате переменная <code>\$result</code> получает ложное значение. <code>\$foo = is_valid('foo'); \$this->assertFalse(\$valid, t('Make sure that the foo is not a valid variable'));</code>
<code>\$this->assertNull(\$value, \$message='%', \$group='Other')</code>	Эта функция утверждает, что в результате переменная <code>\$result</code> получает значение <code>NULL</code> . <code>\$result = load_my_object(-1); \$this->assertNull(\$result, t('Make sure we get NULL when trying to load an invalid object.'));</code>
<code>\$this->assertNotNull(\$value, \$message='%s', \$group='Other')</code>	Эта функция утверждает, что в результате переменная <code>\$result</code> не получает значение <code>NULL</code> .
<code>\$this->assertEqual(\$first, \$second, \$message = '%s', \$group = 'Other')</code>	Эта функция утверждает, что <code>\$first</code> приблизительно эквивалентна (<code>==</code>) <code>\$second</code> .
<code>\$this->assertNotEqual(\$first, \$second, \$message = '%s', \$group='Other')</code>	Эта функция утверждает, что <code>\$first</code> не равна (<code>!=</code>) <code>\$second</code> .
<code>\$this->assertIdentical(\$first, \$second, \$message = '%s', \$group='Other')</code>	Эта функция утверждает, что <code>\$first</code> идентична (<code>===</code>) <code>\$second</code> .
<code>\$this->assertNotIdentical(\$first, \$second, \$message = '%s', \$group='Other')</code>	Эта функция утверждает, что <code>\$first</code> не идентична (<code>!==</code>) <code>\$second</code> .
<code>\$this->assertPattern(\$pattern, \$message = '%s', \$group = 'Other')</code>	Эта функция утверждает, что необработанный HTML-контент текущей страницы соответствует регулярному выражению, определенному в <code>\$pattern</code> .
<code>\$this->assertNoPattern(\$pattern, \$message = '%s', \$group = 'Other')</code>	Эта функция утверждает, что необработанный HTML-контент текущей страницы не соответствует регулярному выражению, определенному в <code>\$pattern</code> .
<code>\$this->assertRaw(\$raw, \$message='% s', \$group='Other')</code>	Эта функция утверждает, что HTML-контент, определенный в параметре <code>\$raw</code> , существует в контенте на текущей странице.
<code>\$this->assertNoRaw(\$raw, \$message='%s', \$group='Other')</code>	Эта функция утверждает, что HTML-контент, определенный в параметре <code>\$raw</code> , не существует в контенте на текущей странице.

Утверждение	Описание
<code>\$this->assertText(\$text, \$message = '%s', \$group='Other')</code>	Эта функция утверждает, что значение, сохраненное в параметре <code>\$text</code> , отображается на текущей странице.
<code>\$this->assertNoText(\$text, \$message = '%s', \$group='Other')</code>	Эта функция утверждает, что значение, сохраненное в параметре <code>\$text</code> , не отображается на текущей странице.
<code>\$this->assertTitle(\$title, \$message = '%s', \$group='Other')</code>	Эта функция утверждает, что заголовок, определенный в параметре <code>\$title</code> , находится на текущей странице.
<code>\$this->assertNoTitle(\$title, \$message = '%s', \$group='Other')</code>	Эта функция утверждает, что заголовок, определенный в параметре <code>\$title</code> , не находится на текущей странице.
<code>\$this->assertUniqueText(\$text, \$message='%s', \$group='Other')</code>	Эта функция утверждает, что текст, определенный в параметре <code>\$text</code> , отображается на текущей странице один и только один раз.
<code>\$this->assertNoUniqueText(\$text, \$message='%s', \$group='Other')</code>	Эта функция утверждает, что текст, определенный в параметре <code>\$text</code> , отображается на текущей странице более одного раза.
<code>\$this->assertLink(\$label, \$index = 0, \$message='%s', \$group='Other')</code>	Эта функция утверждает, что ссылка с указанным текстовым представлением ссылки (<code>\$link</code>) существует на странице. Если на странице существует более одной ссылка с тем же самым текстовым представлением, можно использовать параметр <code>\$index</code> , чтобы определить, какую ссылку требуется протестировать.
<code>\$this->assertNoLink(\$label, \$message='%s', \$group='Other')</code>	Эта функция утверждает, что на странице не существует ни одной ссылки с указанной надписью.
<code>\$this->assertLinkByHref(\$href, \$index=0, \$message='%s', \$group='Other')</code>	Эта функция утверждает, что ссылка с данной <code>\$href</code> или частичной <code>\$href</code> существует на странице. <code>\$this->assertLinkByHref('node/1', 0, 'A link to node 1 appears on the page');</code>
<code>\$this->assertNoLinkByHref(\$href, \$message='%s', \$group='Other')</code>	Эта функция утверждает, что ни одной ссылки с данной <code>\$href</code> или частичной <code>\$href</code> не существует на странице.
<code>\$this->assertResponse(\$code, \$message='%s')</code>	Эта функция утверждает, что код HTTP-ответа для текущей страницы соответствует значению, присвоенному параметру <code>\$code</code> .
<code>\$this->assertFieldById(\$id, \$value=' ', \$message='%s')</code>	Эта функция утверждает, что на текущей странице существует поле с данным идентификатором и значением.
<code>\$this->assertNoFieldById(\$id, \$value=' ', \$message='%s')</code>	Эта функция утверждает, что на текущей странице не существует поле с данным идентификатором и значением.
<code>\$this->assertFieldByName(\$name, \$value=' ', \$message='%s')</code>	Эта функция утверждает, что на текущей странице существует поле с данным именем и значением.

Утверждение	Описание
<code>\$this->assertNoFieldByName(\$name, \$value = ' ', \$message = '%s')</code>	Эта функция утверждает, что на текущей странице не существует поле с данным именем и значением.
<code>\$this->assertFieldChecked(\$id, \$message = '%s')</code>	Эта функция утверждает, что флажок с данным идентификатором существует на текущей странице и отмечен.
<code>\$this->assertNoFieldChecked(\$id, \$message = '%s')</code>	Эта функция утверждает, что флажок с данным идентификатором существует на текущей странице и не отмечен.
<code>\$this->assertOptionSelected(\$id, \$option, \$message = '%s')</code>	Эта функция утверждает, что список выборки с данным идентификатором существует на текущей странице, и при этом указанная опция выбрана.
<code>\$this->assertNoOptionSelected(\$id, \$option, \$message = '%s')</code>	Эта функция утверждает, что список выборки с данным идентификатором существует на текущей странице, и при этом указанная опция не выбрана.
<code>\$this->assertFieldByXPath(\$xpath, \$value, \$message = '%s', \$group = 'Other')</code>	Эта функция утверждает, что на текущей странице существует поле с данным путем XPath.
<code>\$this->assertNoFieldByXPath(\$xpath, \$value, \$message = '%s', \$group = 'Other')</code>	Эта функция утверждает, что на текущей странице не существует поле с заданным XPath.
<code>\$this->assertNoDuplicateIds(\$message = '%s', \$group = 'Other')</code>	Эта функция утверждает, что каждый идентификатор HTML на странице используется только для одного элемента.
<code>\$this->pass(\$message = '%s', \$group = 'Other')</code>	Эта функция делает утверждение, которое всегда положительно.
<code>\$this->fail(\$message = '%s', \$group = 'Other')</code>	Эта функция делает утверждение, которое всегда отрицательно.
<code>\$this->error(\$message = '%s', \$group = 'Other')</code>	Эта функция делает утверждение, которое всегда ведет к условию ошибки.

Резюме

Методики и инструменты, описанные в этой главе, предоставляют все, что необходимо для настройки комплекта повторяемых тестов нового сайта. Хотя процедура может казаться отнимающей много времени, и это действительно так, преимущества затраты времени на документирование и реализацию тестов с применением инструментов, определенных в этой главе, окупят все издержки, когда дело дойдет до внесенных изменений в сайт и необходимости проверки того, что изменение в одном разделе сайта не оказывает пагубного влияния на функциональность где-то в другом месте. Как было продемонстрировано в примере, используя платформу тестирования, удалось выполнить 241 тест в модуле Blog менее чем за 2 минуты — попробуйте проделать это вручную, снова и снова, не выходя их браузера.

ПРИЛОЖЕНИЕ А

Справочник по таблицам базы данных

В этом приложении описаны таблицы базы данных и поля, составляющие ядро Drupal. Описания взяты из реализаций `hook_schema()` в файлах `.install` модулей ядра, с некоторыми минимальными изменениями для ясности. Информация воспроизведена в справочных целях.

Текущие определения таблиц можно найти в файле `.install` установленного в системе Drupal хука схемы того или иного модуля, либо же используя дополнительный модуль `schema`, находящийся на странице <http://drupal.org/project/schema>. Определения таблиц ядра, не относящихся к модулям, приведены в файле `modules/system/system.install`. Если какая-то таблица используется в основном с конкретным модулем, его имя указывается в круглых скобках после имени таблицы. В ссылках на другие таблицы их имена отображаются в фигурных скобках.

На заметку! Столбец типа в следующих таблицах отображает типы данных Drupal, а не типы базы данных. Многие из этих типов непосредственно преобразуются в определения базы данных — например, `varchar` — но к другим это неприменимо вследствие различий между платформами баз данных.

Таблица `accesslog` хранит информацию доступа к сайту и служит статистическим целям.

Таблица А.1. Таблица `accesslog` (модуль `statistics`)

Имя	Тип	Null	По умолчанию	Описание
<code>aid</code>	<code>serial</code>	Нет		Первичный ключ: уникальный идентификатор доступа
<code>sid</code>	<code>varchar(64)</code>	Нет		Идентификатор сеанса браузера пользователя, посетившего страницу
<code>title</code>	<code>varchar(255)</code>	Да		Заголовок посещенной страницы
<code>path</code>	<code>varchar(255)</code>	Да		Внутренний путь к посещенной странице (относительно корня Drupal)
<code>url</code>	<code>varchar(255)</code>	Да		URI источника ссылки
<code>hostname</code>	<code>varchar(128)</code>	Да		Имя хоста пользователя, посетившего страницу

Окончание табл. А.1

Имя	Тип	Null	По умолчанию	Описание
uid	int, unsigned	Да	0	Пользователь {users}.uid, посетивший страницу
timer	int, unsigned	Нет	0	Время в миллисекундах, которое потребовалось для загрузки страницы
timestamp	int, unsigned	Нет	0	Временная метка момента посещения страницы

Таблица actions хранит информацию о действиях.

Таблица А.2. Таблица actions

Имя	Тип	Null	По умолчанию	Описание
aid	varchar(255)	Нет		Первичный ключ: уникальный идентификатор действия
type	varchar(32)	Нет		Объект, над которым производится действие (нода, пользователь, комментарий, система или нестандартные типы)
callback	varchar(255)	Нет		Функция обратного вызова, исполняемая при запуске действия
parameters	longblob	Нет		Параметры, передаваемые функции обратного вызова
description	varchar(255)	Нет		Описание действия

Таблица aggregator_category хранит категории для каналов агрегатора и элементы каналов.

Таблица А.3. Таблица aggregator_category (модуль aggregator)

Имя	Тип	Null	По умолчанию	Описание
cid	serial	Нет		Первичный ключ: уникальный идентификатор категории агрегатора
title	varchar(255)	Нет		Заголовок категории
description	text:big	Нет		Описание категории
block	int:tiny	Нет	0	Количество последних элементов, которые нужно отображать внутри блока категории

Таблица aggregator_category_feed отображает каналы на категории.

Таблица А.4. Таблица aggregator_category_feed (модуль aggregator)

Имя	Тип	Null	По умолчанию	Описание
fid	int	Нет	0	{aggregator_feed}.fid канала
cid	int	Нет	0	{aggregator_category}.cid, которому назначен канал

Таблица `aggregator_category_item` отображает элементы каналов категориями.

Таблица А.5. Таблица `aggregator_category_item` (модуль `aggregator`)

Имя	Тип	Null	По умолчанию	Описание
<code>iid</code>	<code>int</code>	Нет	0	<code>{aggregator_item}.iid</code> элемента канала
<code>cid</code>	<code>int</code>	Нет	0	<code>{aggregator_category}.cid</code> , которому назначен элемент канала

Таблица `aggregator_feed` хранит каналы, которые должны быть проанализированы агрегатором.

Таблица А.6. Таблица `aggregator_feed` (модуль `aggregator`)

Имя	Тип	Null	По умолчанию	Описание
<code>fid</code>	<code>serial</code>	Нет		Первичный ключ: уникальный идентификатор канала
<code>title</code>	<code>varchar(255)</code>	Нет		Заголовок канала
<code>url</code>	<code>varchar(255)</code>	Нет		URL канала
<code>refresh</code>	<code>int</code>	Нет	0	Частота проверки новых элементов канала, в секундах
<code>checked</code>	<code>int</code>	Нет	0	Временная метка Unix последней проверки канала на наличие новых элементов
<code>queued</code>	<code>int</code>	Нет	0	Время помещения данного канала в очередь на обновление; 0, если канал не помещен в очередь
<code>link</code>	<code>varchar(255)</code>	Нет		Родительский веб-сайт канала; поступает из элемента <code><link></code> канала
<code>description</code>	<code>text:big</code>	Нет		Описание родительского веб-сайта; поступает из элемента <code><description></code> канала
<code>image</code>	<code>text:big</code>	Нет		Изображение, представляющее канал
<code>hash</code>	<code>varchar(64)</code>	Нет		Вычисленное хеш-значение данных канала; используется для проверки достоверности кэша
<code>etag</code>	<code>varchar(255)</code>	Нет	' '	Заголовок HTTP-ответа дескриптора объекта; используется для проверки достоверности кэша
<code>modified</code>	<code>int</code>	Нет	0	Временная метка Unix последней модификации канала
<code>block</code>	<code>int:tiny</code>	Нет	0	Количество элементов для отображения в блоке канала

500 Приложение А. Справочник по таблицам базы данных

Таблица `aggregator_item` хранит индивидуальные элементы, импортированные из канала.

Таблица А.7. Таблица `aggregator_item` (модуль `aggregator`)

Имя	Тип	Null	По умолчанию	Описание
<code>iid</code>	<code>serial</code>	Нет		Первичный ключ: уникальный идентификатор элемента канала
<code>fid</code>	<code>int</code>	Нет	0	<code>{aggregator_feed}.fid</code> , которому принадлежит этот элемент
<code>title</code>	<code>varchar(255)</code>	Нет	' '	Заголовок элемента канала
<code>link</code>	<code>varchar(255)</code>	Нет	' '	Ссылка на элемент канала
<code>author</code>	<code>varchar(255)</code>	Нет	' '	Автор элемента канала
<code>description</code>	<code>text:big</code>	Нет		Тело элемента канала
<code>timestamp</code>	<code>int</code>	Да		Временная метка Unix отправки элемента канала
<code>guid</code>	<code>varchar(255)</code>	Да		Уникальный идентификатор элемента канала

Таблица `authmap` хранит информацию отображения распределенной аутентификации.

Таблица А.8. Таблица `authmap` (модуль `user`)

Имя	Тип	Null	По умолчанию	Описание
<code>aid</code>	<code>serial</code>	Нет		Первичный ключ: уникальный идентификатор отображения аутентификации
<code>uid</code>	<code>int</code>	Нет	0	<code>{users}.uid</code> пользователя
<code>authname</code>	<code>varchar(128)</code>	Нет	' '	Уникальное имя аутентификации
<code>module</code>	<code>varchar(128)</code>	Нет	' '	Модуль, управляющий аутентификацией

Таблица `batch` хранит детальную информацию о пакетах (процессах, выполняющихся в виде множества HTTP-запросов).

Таблица А.9. Таблица `batch` (`batch.inc`)

Имя	Тип	Null	По умолчанию	Описание
<code>bid</code>	<code>serial</code>	Нет		Первичный ключ: уникальный идентификатор пакета
<code>token</code>	<code>varchar(64)</code>	Нет		Строковый маркер, сгенерированный по идентификатору сеанса текущего пользователя и идентификатору пакета; служит для гарантии того, что только пользователь, отправивший пакет, может эффективно обращаться к нему
<code>timestamp</code>	<code>int</code>	Нет		Временная метка Unix, указывающая время отправки данного пакета для обработки; устаревшие пакеты удаляются во время запуска <code>cron</code>
<code>batch</code>	<code>text:big</code>	Да		Сериализованный массив, содержащий данные для обработки пакетом

Таблица `blocks` хранит настройки блока, такие как настройки области и видимости.

Таблица А.10. Таблица `blocks` (модуль `block`)

Имя	Тип	Null	По умолчанию	Описание
<code>bid</code>	<code>serial</code>	Нет		Первичный ключ: уникальный идентификатор блока
<code>module</code>	<code>varchar(64)</code>	Нет	<code>' '</code>	Модуль, из которого происходит блок; например, <code>user</code> для блока Who's Online или <code>block</code> для любого нестандартного блока
<code>delta</code>	<code>varchar(32)</code>	Нет	<code>'0'</code>	Уникальный идентификатор блока внутри модуля
<code>theme</code>	<code>varchar(64)</code>	Нет	<code>' '</code>	Тема, в рамках которой применяются установки блока
<code>status</code>	<code>int:tiny</code>	Нет	<code>0</code>	Состояние включения блока (1 — активирован, 0 — нет)
<code>weight</code>	<code>int:tiny</code>	Нет	<code>0</code>	Вес блока внутри области
<code>region</code>	<code>varchar(64)</code>	Нет	<code>' '</code>	Область темы, внутри которой установлен блок
<code>custom</code>	<code>int:tiny</code>	Нет	<code>0</code>	Флаг, указывающий, как пользователи могут управлять видимостью блока; 0 указывает, что пользователи не могут управлять ею; 1 означает, что по умолчанию блок активирован, но может быть скрыт; 2 означает, что по умолчанию блок скрыт, но может быть показан
<code>visibility</code>	<code>int:tiny</code>	Нет	<code>0</code>	Флаг, указывающий способ отображения блоков на страницах. 0 означает отображение на всех страницах, кроме специально перечисленных; 1 означает отображение только на перечисленных страницах; 2 означает необходимость использования специального PHP-кода для определения видимости
<code>pages</code>	<code>text</code>	Нет		Контент блока <code>Pages</code> (Страницы); содержит либо список путей, по которым можно включить либо исключить блок или PHP-код в зависимости от настройки <code>visibility</code>
<code>title</code>	<code>varchar(64)</code>	Нет	<code>' '</code>	Нестандартный заголовок блока. Пустая строка приведет к использованию заголовка по умолчанию, <code><none></code> — заголовок будет удален, некоторый текст — блок будет использовать этот текст в качестве заголовка
<code>cache</code>	<code>int:tiny</code>	Нет	<code>1</code>	Двоичный флаг, указывающий режим кэширования блока; -1 означает запрещение кэширования; 1 означает создание одного кэша для каждой роли; 2 означает создание кэша для каждого пользователя, 4 — одного кэша на страницу, 8 — кэш блока является глобальным. Режимы кэширования блоков рассматривались в главе 10

Таблица `blocked_ips` хранит список заблокированных IP-адресов.

Таблица А.11. Таблица `blocked_ips`

Имя	Тип	Null	По умолчанию	Описание
<code>iid</code>	<code>serial</code>	Нет		Первичный ключ: уникальный идентификатор IP-адреса
<code>ip</code>	<code>varchar(40)</code>	Нет		IP-адрес

Таблица `block_custom` хранит содержимое специально созданных блоков.

Таблица А.12. Таблица `block_custom` (модуль `block`)

Имя	Тип	Null	По умолчанию	Описание
<code>bid</code>	<code>serial, unsigned</code>	Нет		<code>{block}.bid</code> блока
<code>body</code>	<code>text:big</code>	Да	Null	Содержимое блока
<code>info</code>	<code>varchar(128)</code>	Нет		Описание блока
<code>format</code>	<code>int:small</code>	Нет	0	<code>{filter_format}.format</code> тела блока

Таблица `blocks_node_type` хранит информацию, которая устанавливает критерии отображения блоков в зависимости от типа их контента.

Таблица А.13. Таблица `blocks_node_type` (модуль `block`)

Имя	Тип	Null	По умолчанию	Описание
<code>module</code>	<code>varchar(64)</code>	Нет		Исходный модуль блока, из <code>{block}.module</code>
<code>delta</code>	<code>varchar(32)</code>	Нет		Уникальное смещение блока в модуле, из <code>{block}.delta</code>
<code>type</code>	<code>varchar(32)</code>	Нет		Машиночитаемое имя этого типа, из <code>{node_type}.type</code>

Таблица `blocks_role` хранит права доступа для блоков на основе пользовательских ролей.

Таблица А.14. Таблица `blocks_role` (модуль `block`)

Имя	Тип	Null	По умолчанию	Описание
<code>module</code>	<code>varchar(64)</code>	Нет		Исходный модуль блока, из <code>{blocks}.module</code>
<code>delta</code>	<code>varchar(32)</code>	Нет		Уникальное смещение блока в модуле, из <code>{blocks}.delta</code>
<code>rid</code>	<code>int, unsigned</code>	Нет		Идентификатор роли пользователя, из <code>{users_roles}.rid</code>

Таблица `book` хранит описательную информацию о книге и связывает каждую ноду в описании с уникальной ссылкой в таблице `menu_links`.

Таблица А.15. Таблица `book` (модуль `book`)

Имя	Тип	Null	По умолчанию	Описание
<code>mlid</code>	<code>int, unsigned</code>	Нет	0	<code>{menu_links}.mlid</code> страницы книги
<code>nid</code>	<code>int, unsigned</code>	Нет	0	<code>{node}.nid</code> страницы книги
<code>bid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор книги — <code>{book}.nid</code> страницы верхнего уровня

Таблица `cache` используется для кэширования элементов, не разделенных по собственным таблицам кэширования. Дополнительные модули также могут применять эту таблицу для хранения своих кэшированных элементов.

Таблица А.16. Таблица `cache`

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет	' '	Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	0	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_block` модуля `block` предназначена для хранения уже построенных блоков, идентифицируемых по модулю, смещению и различным контекстам, которые могут изменять блок, таким как тема, локаль и режим кэширования, определенные для блока.

Таблица А.17. Таблица `cache_block` (модуль `block`)

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет	' '	Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	0	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_bootstrap` хранит данные начальной загрузки кэша.

Таблица A.18. Таблица `cache_bootstrap`

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет	<code>' '</code>	Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	<code>0</code>	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	<code>0</code>	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	<code>0</code>	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_field` хранит кэшируемые значения полей.

Таблица A.19. Таблица `cache_field`

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет	<code>' '</code>	Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	<code>0</code>	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	<code>0</code>	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	<code>0</code>	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_filter` хранит уже отфильтрованные фрагменты текста, которые идентифицированы по формату ввода и хешу sha-256 текста.

Таблица A.20. Таблица `cache_filter` (модуль `filter`)

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет	<code>' '</code>	Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	<code>0</code>	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	<code>0</code>	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	<code>0</code>	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_form` хранит недавно построенные формы и их хранимые данные для использования в последующих запросах страниц.

Таблица A.21. Таблица `cache_form`

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет		Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	0	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_image` используется для хранения информации о манипуляциях с изображениями, находящихся в стадии выполнения.

Таблица A.22. Таблица `cache_image`

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет	' '	Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	0	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_menu` хранит информацию маршрутизатора наряду со сгенерированными деревьями ссылок для различных комбинаций меню/страница/пользователь.

Таблица A.23. Таблица `cache_menu`

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет	' '	Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	0	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_page` используется для хранения сжатых страниц для анонимных пользователей, если кэширование страниц включено.

Таблица A.24. Таблица `cache_page`

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет		Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	0	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_path` хранит псевдонимы пути.

Таблица A.25. Таблица `cache_path`

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет		Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	0	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `cache_update` хранит загруженную из `drupal.org` информацию о доступных выпусках ядра, модулей, и тем Drupal, загруженных на данном сайте.

Таблица A.26. Таблица `cache_update`

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>varchar(255)</code>	Нет		Первичный ключ: уникальный идентификатор кэша
<code>data</code>	<code>blob:big</code>	Да		Коллекция данных для кэширования
<code>expire</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая срок устаревания записи кэша. Равна 0, если срок существования записи не ограничен
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая время создания записи кэша
<code>serialized</code>	<code>int:small</code>	Нет	0	Флаг — признак сериализации содержимого (да — 1, нет — 0)

Таблица `comment` хранит комментарии и связанные с ними данные.

Таблица А.27. Таблица `comment` (модуль `comment`)

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>serial</code>	Нет		Первичный ключ: уникальный идентификатор комментария
<code>pid</code>	<code>int</code>	Нет	0	<code>{comments}.cid</code> , которому данный комментарий служит ответом. Если это значение установлено в 0, данный комментарий не является ответом на существующий комментарий
<code>nid</code>	<code>int</code>	Нет	0	<code>{node}.nid</code> , которому данный комментарий является ответом
<code>uid</code>	<code>int</code>	Нет	0	<code>{users}.uid</code> , представляющий автора данного комментария. Если данное значение установлено в 0, комментарий был создан анонимным пользователем
<code>subject</code>	<code>varchar(64)</code>	Нет	' '	Заголовок комментария
<code>hostname</code>	<code>varchar(128)</code>	Нет	' '	Имя хоста автора
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix момента создания комментария
<code>changed</code>	<code>int</code>	Нет	0	Временная метка Unix момента обновления комментария
<code>status</code>	<code>int:tiny, unsigned</code>	Нет	1	Состояние публикации комментария (0 означает "опубликован", 1 — "не опубликован")
<code>thread</code>	<code>varchar(255)</code>	Нет		<code>vanocode</code> -представление места комментария в цепочке
<code>name</code>	<code>varchar(60)</code>	Да		Имя автора комментария. Использует <code>{users}.name</code> , если пользователь зарегистрирован в системе; в противном случае использует значение, введенное в форме комментария
<code>mail</code>	<code>varchar(64)</code>	Да		Адрес электронной почты автора комментария из формы комментария, если пользователь анонимный и включена настройка <code>Anonymous users may/must leave their contact information</code> (Анонимные пользователи могут/должны оставлять свою контактную информацию)
<code>homepage</code>	<code>varchar(255)</code>	Да		Адрес домашней страницы пользователя из формы комментария, если пользователь анонимный и включена настройка <code>Anonymous users may/must leave their contact information</code>
<code>language</code>	<code>varchar(12)</code>	Нет		Язык, на котором был создан комментарий

Таблица `contact` хранит настройки категорий формы контактов.

Таблица А.28. Таблица `contact` (модуль `contact`)

Имя	Тип	Null	По умолчанию	Описание
<code>cid</code>	<code>serial, unsigned</code>	Нет		Первичный ключ: уникальный идентификатор категории
<code>category</code>	<code>varchar(255)</code>	Нет		Имя категории
<code>recipients</code>	<code>text:big</code>	Нет		Разделенный запятыми список адресов электронной почты получателей
<code>reply</code>	<code>text:big</code>	Нет		Текст автоматического ответного сообщения
<code>weight</code>	<code>int</code>	Нет	0	Вес категории
<code>selected</code>	<code>int:tiny</code>	Нет	0	Флаг, указывающий, выбрана ли категория по умолчанию (1 — да, 0 — нет)

Модуль `date_formats` хранит сконфигурированные форматы даты.

Таблица А.29. Таблица `date_formats`

Имя	Тип	Null	По умолчанию	Описание
<code>dfid</code>	<code>serial, unsigned</code>	Нет		Идентификатор формата даты
<code>format</code>	<code>varchar(100)</code>	Нет		Строка формата даты
<code>type</code>	<code>varchar(64)</code>	Нет		Тип формата даты, например, <code>medium</code> (средний)
<code>locked</code>	<code>int:tiny</code>	Нет	0	Может ли этот формат модифицироваться

Таблица `date_format_locale` хранит сконфигурированные форматы даты для каждой локали.

Таблица А.30. Таблица `date_format_locale`

Имя	Тип	Null	По умолчанию	Описание
<code>format</code>	<code>varchar(100)</code>	Нет		Строка формата даты
<code>type</code>	<code>varchar(64)</code>	Нет		Тип формата даты, например, <code>medium</code> (средний)
<code>language</code>	<code>varchar(12)</code>	Нет		<code>{languages}.language</code> , с которым должен использоваться этот формат

Таблица `date_format_type` хранит типы сконфигурированного формата даты.

Таблица А.31. Таблица `date_format_type`

Имя	Тип	Null	По умолчанию	Описание
<code>type</code>	<code>varchar(64)</code>	Нет		Тип формата даты, например, <code>medium</code>
<code>title</code>	<code>varchar(255)</code>	Нет		Удобочитаемое имя типа формата
<code>locked</code>	<code>int:tiny</code>	Нет	0	Является ли этот формат предоставленным системой

Таблица `field_config` хранит информацию о конфигурации полей.

Таблица A.32. Таблица `field_config`

Имя	Тип	Null	По умолчанию	Описание
<code>id</code>	<code>serial</code>	Нет		Первичный идентификатор поля
<code>field_name</code>	<code>varchar(32)</code>	Нет		Имя этого поля; имена неудаленных полей являются уникальными, но многократно удаленные поля могут иметь одно и то же имя
<code>type</code>	<code>varchar(128)</code>	Нет		Тип данного поля
<code>module</code>	<code>varchar(128)</code>	Нет		Модуль, реализующий тип поля
<code>active</code>	<code>int:tiny</code>	Нет	0	Булевское значение, указывающее, включен ли модуль, реализующий тип поля
<code>storage_type</code>	<code>varchar(128)</code>	Нет		Серверная база данных хранилища для поля
<code>storage_module</code>	<code>varchar(128)</code>	Нет		Модуль, реализующий серверную базу данных хранилища
<code>storage_active</code>	<code>int:tiny</code>	Нет	0	Булевское значение, указывающее, включен ли модуль, реализующий серверную базу данных
<code>locked</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое определяет, может ли поле модифицироваться
<code>data</code>	<code>longblob</code>	Нет		Сериализованные данные, содержащие свойства полей, которые не удостоверяют выделенный столбец
<code>cardinality</code>	<code>int:tiny</code>	Нет	0	Кардинальность поля
<code>translatable</code>	<code>int:tiny</code>	Нет	0	Определяет, переводимо ли поле
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое определяет, было ли поле удалено

Таблица `field_config_entity_type` хранит информацию об объекте, которая используется API-интерфейсом работы с полями.

Таблица A.33. Таблица `field_config_entity_type`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial, unsigned</code>	Нет		Уникальный идентификатор для этого типа объекта
<code>type</code>	<code>varchar(128)</code>	Нет		Тип объекта

510 Приложение А. Справочник по таблицам базы данных

Таблица `field_config_instance` хранит информацию о конфигурации полей.

Таблица A.34. Таблица `field_config_instance`

Имя	Тип	Null	По умолчанию	Описание
<code>id</code>	<code>serial</code>	Нет		Первичный идентификатор экземпляра поля
<code>field_id</code>	<code>int</code>	Нет		Идентификатор поля, присоединенного посредством этого экземпляра
<code>field_name</code>	<code>varchar(32)</code>	Нет		Имя поля
<code>entity_type</code>	<code>varchar(32)</code>	Нет		Тип объекта, связанного с этой конфигурацией
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, с которым связана эта конфигурация
<code>data</code>	<code>blob:big</code>	Нет		Хранит содержимое поля
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое идентифицирует, была ли удалена конфигурация этого поля

Таблица `field_data_body` хранит сведения о поле тела объекта.

Таблица A.35. Таблица `field_data_body`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial, unsigned</code>	Нет		Идентификатор объекта
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, связанный с этим объектом
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли запись удалена
<code>entity_id</code>	<code>int, unsigned</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int, unsigned</code>	Да	NULL	Идентификатор версии
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с этим объектом
<code>delta</code>	<code>int, unsigned</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>body_value</code>	<code>text:big</code>	Да	NULL	Контент, хранимый в теле
<code>body_summary</code>	<code>text:big</code>	Да	NULL	Краткая сводка по контенту, хранимому в теле
<code>body_format</code>	<code>int, unsigned</code>	Да	NULL	Формат контента, хранимого в теле

Таблица `field_data_comment_body` хранит информацию о комментариях, связанных с объектом.

Таблица A.36. Таблица `field_data_comment_body`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial, unsigned</code>	Нет		Идентификатор объекта
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, связанный с этим объектом
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли запись удалена
<code>entity_id</code>	<code>int</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int</code>	Да	NULL	Идентификатор версии
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с этим объектом
<code>delta</code>	<code>int</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>comment_body_value</code>	<code>text:big</code>	Да	NULL	Содержимое тела комментария
<code>comment_body_format</code>	<code>int</code>	Да	NULL	Формат тела комментария

Таблица `field_data_field_image` хранит информацию об изображениях, связанных с объектом.

Таблица A.37. Таблица `field_data_field_image`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial</code>	Нет		Идентификатор объекта
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, связанный с этим объектом
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли запись удалена
<code>entity_id</code>	<code>int</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int</code>	Да	NULL	Идентификатор версии
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с этим объектом
<code>delta</code>	<code>int</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>field_image_fid</code>	<code>int</code>	Да	NULL	Идентификатор файла изображения
<code>field_image_alt</code>	<code>varchar(128)</code>	Да	NULL	Дескриптор ALT, связанный с изображением
<code>field_image_title</code>	<code>varchar(128)</code>	Да	NULL	Заголовок, связанный с изображением

512 Приложение А. Справочник по таблицам базы данных

Таблица `field_data_field_tags` хранит информацию о дескрипторах, связанных с объектом.

Таблица А.38. Таблица `field_data_field_tags`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial</code>	Нет		Идентификатор объекта
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, связанный с этим объектом
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли запись удалена
<code>entity_id</code>	<code>int</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int</code>	Да	NULL	Идентификатор версии
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с этим объектом
<code>delta</code>	<code>int</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>field_tags_tid</code>	<code>int</code>	Да	NULL	Идентификатор термина таксономии, связанного с дескрипторами объекта

Таблица `field_data_taxonomy_forums` хранит информацию о терминах таксономии, связанных с форумами.

Таблица А.39. Таблица `field_data_taxonomy_forums`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial</code>	Нет		Идентификатор объекта
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, связанный с этим объектом
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли запись удалена
<code>entity_id</code>	<code>int</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int</code>	Да	NULL	Идентификатор версии
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с объектом
<code>delta</code>	<code>int</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>taxonomy_forums_tid</code>	<code>int</code>	Да	NULL	Идентификатор термина, связанного с форумом

Таблица `field_revision_body` хранит сведения о версиях полей тела.

Таблица A.40. Таблица `field_revision_body`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial, unsigned</code>	Нет		Идентификатор объекта
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, связанный с этим объектом
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли эта запись удалена
<code>entity_id</code>	<code>int</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int</code>	Да	NULL	Идентификатор версии
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с этим объектом
<code>delta</code>	<code>int</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>body_value</code>	<code>text:big</code>	Да	NULL	Значение, сохраненное в теле объекта
<code>body_summary</code>	<code>text:big</code>	Да	NULL	Сводка по тексту тела
<code>body_format</code>	<code>int</code>	Да	NULL	Формат текста тела

Таблица `field_revision_comment_body` хранит информацию о версиях комментариев.

Таблица A.41. Таблица `field_revision_comment_body`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial, unsigned</code>	Нет		Идентификатор типа объекта, к которому присоединено поле
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, связанный с этим объектом
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли эта запись удалена
<code>entity_id</code>	<code>int</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int</code>	Да	NULL	Идентификатор версии объекта, к которому присоединены эти данные
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с этим объектом
<code>delta</code>	<code>int, unsigned</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>comment_body_value</code>	<code>text:big</code>	Да	NULL	Значение, связанное с телом комментария
<code>comment_body_format</code>	<code>int, unsigned</code>	Да	NULL	Формат, связанный с телом комментария

514 Приложение А. Справочник по таблицам базы данных

Таблица `field_revision_field_image` хранит сведения о версиях изображений.

Таблица А.42. Таблица `field_revision_field_image`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial</code>	Нет		Идентификатор объекта
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, связанный с этим объектом
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли эта запись удалена
<code>entity_id</code>	<code>int</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int</code>	Да	NULL	Идентификатор версии
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с этим объектом
<code>delta</code>	<code>int, unsigned</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>field_image_fid</code>	<code>int, unsigned</code>	Да	NULL	Идентификатор файла изображения
<code>field_image_alt</code>	<code>varchar(128)</code>	Да	NULL	Дескриптор ALT, связанный с изображением
<code>field_image_title</code>	<code>varchar(128)</code>	Да	NULL	Заголовок, связанный с этим изображением

Таблица `field_revision_field_tags` хранит информацию о версиях терминов/тегов таксономии, связанных с объектом.

Таблица А.43. Таблица `field_revision_field_tags`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial, unsigned</code>	Нет		Идентификатор объекта
<code>bundle</code>	<code>varchar(128)</code>	Нет		Пакет, связанный с этим объектом
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли эта запись удалена
<code>entity_id</code>	<code>int, unsigned</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int, unsigned</code>	Да	NULL	Идентификатор версии
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с этим объектом
<code>delta</code>	<code>int, unsigned</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>field_tags_tid</code>	<code>int</code>	Да	NULL	Идентификатор термина связанных тегов

Таблица `field_revision_taxonomy_forums` хранит информацию о версиях терминов таксономии, связанных с форумами.

Таблица А.44. Таблица `field_revision_taxonomy_forums`

Имя	Тип	Null	По умолчанию	Описание
<code>etid</code>	<code>serial, unsigned</code>	Нет		Идентификатор объекта
<code>bundle</code>	<code>varchar(128)</code>	Нет		Присоединенный пакет
<code>deleted</code>	<code>int:tiny</code>	Нет	0	Булевское значение, которое указывает, была ли запись удалена
<code>entity_id</code>	<code>int, unsigned</code>	Нет		Идентификатор связанного объекта
<code>revision_id</code>	<code>int, unsigned</code>	Да	NULL	Идентификатор версии
<code>language</code>	<code>varchar(32)</code>	Нет		Язык, связанный с этим экземпляром
<code>delta</code>	<code>int, unsigned</code>	Нет		Порядковый номер этого элемента данных, используемый для полей, которые содержат несколько значений
<code>taxonomy_forums_tid</code>	<code>int, unsigned</code>	Да	NULL	Идентификатор термина форума, связанного с этим объектом

Таблица `file_managed` хранит информацию о выгруженных файлах.

Таблица А.45. Таблица `file_managed`

Имя	Тип	Null	По умолчанию	Описание
<code>fid</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный ключ: уникальный идентификатор файла
<code>uid</code>	<code>int, unsigned</code>	Нет	0	<code>{users}.uid</code> пользователя, ассоциированного с файлом
<code>filename</code>	<code>varchar(255)</code>	Нет		Имя файла
<code>uri</code>	<code>varchar(255)</code>	Нет		Путь файла относительно корня Drupal
<code>filemime</code>	<code>varchar(255)</code>	Нет		MIME-тип файла
<code>filesize</code>	<code>int, unsigned</code>	Нет	0	Размер файла в байтах
<code>status</code>	<code>int</code>	Нет	0	Флаг, указывающий, является файл временным (0) или постоянным (1)
<code>timestamp</code>	<code>int, unsigned</code>	Нет	0	Временная метка Unix, указывающая время добавления файла

516 Приложение А. Справочник по таблицам базы данных

Таблица `file_usage` хранит информацию для отслеживания места использования файла.

Таблица A.46. Таблица `file_usage` (модуль `system`)

Имя	Тип	Null	По умолчанию	Описание
<code>fid</code>	<code>int.unsigned</code>	Нет		Идентификатор файла отслеживаемого поля
<code>module</code>	<code>varchar(255)</code>	Нет	<code>NULL</code>	Имя модуля, использующего файл
<code>type</code>	<code>varchar(64)</code>	Нет	<code>NULL</code>	Имя типа объекта, в котором используется файл
<code>id</code>	<code>int, unsigned</code>	Нет	<code>0</code>	Первичный ключ объекта, использующего файл
<code>count</code>	<code>int, unsigned</code>	Нет	<code>0</code>	Количество использований данного файла этим объектом

Таблица `filter` отображает фильтры (корректор HTML) на текстовые форматы (Filtered HTML (Фильтрованный HTML)).

Таблица A.47. Таблица `filter` (модуль `filter`)

Имя	Тип	Null	По умолчанию	Описание
<code>format</code>	<code>int</code>	Нет	<code>0</code>	<code>{filter_formats}.format</code> , которому назначен этот фильтр
<code>module</code>	<code>varchar(64)</code>	Нет		Исходный модуль фильтра
<code>name</code>	<code>varchar(32)</code>	Нет		Идентификатор для идентификации фильтра внутри модуля, на который установлена ссылка
<code>weight</code>	<code>int</code>	Нет	<code>0</code>	Вес фильтра внутри формата
<code>status</code>	<code>int</code>	Нет	<code>0</code>	Состояние включения фильтра (1 — активизирован, 0 — нет)
<code>settings</code>	<code>longblob</code>	Да	<code>NULL</code>	Сериализованный массив пар “имя/значение”, который хранит настройки фильтра для конкретного формата

Таблица `filter_format` хранит форматы ввода, представляющие собой специальные группы фильтров, такие как Filtered HTML (Фильтрованный HTML).

Таблица A.48. Таблица `filter_format` (модуль `filter`)

Имя	Тип	Null	По умолчанию	Описание
<code>format</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный ключ: уникальный идентификатор формата
<code>name</code>	<code>varchar(255)</code>	Нет		Имя формата ввода (например, Filtered HTML)
<code>cache</code>	<code>int:tiny</code>	Нет	<code>0</code>	Флаг — признак кэшируемости формата (1 — кэшируемый, 0 — нет)
<code>weight</code>	<code>int:tiny</code>	Нет	<code>0</code>	Вес фильтра внутри формата

Таблица `flood` управляет порогами срабатывания событий, такими как количество попыток подключения.

Таблица А.49. Таблица `flood`

Имя	Тип	Null	По умолчанию	Описание
<code>fid</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный ключ: уникальный идентификатор события лавинного потока
<code>event</code>	<code>varchar(64)</code>	Нет		Имя события (например, <code>contact</code> (контакт))
<code>identifier</code>	<code>varchar(128)</code>	Нет		Идентификатор посетителя, такого как IP-адрес или имя хоста
<code>timestamp</code>	<code>int</code>	Нет	0	Временная метка события
<code>expiration</code>	<code>int</code>	Нет	0	Временная метка срока истечения; события с истекшим сроком удаляются при запуске <code>cron</code>

Таблица `forum` хранит отношения нод к терминам форума.

Таблица А.50. Таблица `forum` (модуль `forum`)

Имя	Тип	Null	По умолчанию	Описание
<code>nid</code>	<code>int, unsigned</code>	Нет	0	<code>{node}.nid</code> ноды
<code>vid</code>	<code>int, unsigned</code>	Нет	0	Первичный ключ: <code>{node}.vid</code> ноды
<code>tid</code>	<code>int, unsigned</code>	Нет	0	<code>{term_data}.tid</code> термина форума, назначенного ноде

Таблица `forum_index` поддерживает денормализованную информацию об отношениях ноды/термин.

Таблица А.51. Таблица `forum_index` (модуль `forum`)

Имя	Тип	Null	По умолчанию	Описание
<code>nid</code>	<code>int, unsigned</code>	Нет	0	<code>{node}.nid</code> , отслеживаемый этой записью
<code>title</code>	<code>varchar(255)</code>	Нет		Заголовок этой ноды, всегда трактуемый как простой неразмеченный текст
<code>tid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор термина
<code>sticky</code>	<code>int:tiny</code>	Да	0	Булевское значение, указывающее, является ли нода закрепленной
<code>created</code>	<code>int, unsigned</code>	Нет	0	Временная метка Unix момента создания ноды
<code>last_comment_timestamp</code>	<code>int</code>	Нет	0	Временная метка Unix последнего комментария, представленного внутри данной ноды, которая получена из <code>{comment}.timestamp</code>
<code>comment_count</code>	<code>int, unsigned</code>	Нет	0	Общее количество комментариев в данной ноды

518 Приложение А. Справочник по таблицам базы данных

Таблица `history` хранит записи об истории чтения нод конкретными пользователями.

Таблица A.52. Таблица `history` (модуль `node`)

Имя	Тип	Null	По умолчанию	Описание
<code>uid</code>	<code>int</code>	Нет	0	Пользователь <code>{users}.uid</code> , читавший ноду <code>{node}.nid</code>
<code>nid</code>	<code>int</code>	Нет	0	Прочитанная нода <code>{node}.nid</code>
<code>timestamp</code>	<code>int</code>	Нет	0	Временная метка Unix момента чтения

Таблица `image_effects` хранит параметры конфигурации для эффектов изображения.

Таблица A.53. Таблица `image_effects`

Имя	Тип	Null	По умолчанию	Описание
<code>ieid</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный идентификатор эффекта изображения
<code>isid</code>	<code>int, unsigned</code>	Нет	0	<code>{image_styles}.isid</code> стиля изображения
<code>weight</code>	<code>int</code>	Нет	0	Вес эффекта в стиле
<code>name</code>	<code>varchar(255)</code>	Нет		Уникальное имя эффекта, который должен быть задействован
<code>data</code>	<code>longblob</code>	Нет		Данные конфигурации эффекта

Таблица `mage_styles` хранит параметры конфигурации стилей изображения.

Таблица A.54. Таблица `image_styles`

Имя	Тип	Null	По умолчанию	Описание
<code>isid</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный идентификатор стиля изображения
<code>name</code>	<code>varchar(255)</code>	Нет		Имя стиля

Таблица `languages` хранит список всех языков, доступных в системе.

Таблица A.55. Таблица `languages` (модуль `locale`)

Имя	Тип	Null	По умолчанию	Описание
<code>language</code>	<code>varchar(128)</code>	Нет		Код языка, например, <code>de</code> или <code>en-US</code>
<code>name</code>	<code>varchar(64)</code>	Нет		Название языка на английском языке
<code>native</code>	<code>varchar(64)</code>	Нет		Родное название языка
<code>direction</code>	<code>int</code>	Нет	0	Направления письма языка (0 — слева направо, 1 — справа налево)
<code>enabled</code>	<code>int</code>	Нет	0	Флаг включения (1 — включен, 0 — отключен)

Окончание табл. А.55

Имя	Тип	Null	По умолчанию	Описание
plurals	int	Нет	0	Количество индексов множественного числа в данном языке
formula	varchar(128)	Нет		Формула множественного числа в РНР-коде для получения индексов множественного числа
domain	varchar(128)	Нет		Домен использования этого языка
prefix	varchar(128)	Нет		Префикс пути для использования с этим языком
weight	int	Нет	0	Вес, используемый в списках языков
javascript	varchar(32)	Нет		Местоположение JavaScript-файла перевода

Таблица `locales_source` хранит список исходных строк на английском языке.

Таблица А.56. Таблица `locales_source` (модуль `locale`)

Имя	Тип	Null	По умолчанию	Описание
lid	serial, unsigned autoincrement	Нет		Уникальный идентификатор данной строки
location	varchar(255)	Нет		Путь Drupal в случае обнаружения онлайн-переводов или путь файла импортируемых строк
texgroup	varchar(255)	Нет	'default'	Определенная модулем группа переводов; см. <code>hook_locale()</code>
source	blob	Нет		Исходная строка на английском языке
context	varchar(255)	Нет		Контекст, к которому применяется эта строка
version	varchar(20)	Нет	'none'	Версия Drupal, в которой последний раз была использована строка (для оптимизации локалей)

Таблица `locales_target` хранит переведенные версии строк.

Таблица А.57. Таблица `locales_target` (модуль `locale`)

Имя	Тип	Null	По умолчанию	Описание
lid	int	Нет	' '	Идентификатор исходной строки, <code>{locales_source}.lid</code> ссылок
translation	blob	Нет		Значение строки перевода на данном языке
language	varchar(12)	Нет		Код языка, <code>{languages}.language</code>
plid	int	Нет	0	Родительский lid (lid предыдущей строки в множественной цепочке) в случае множественных строк, <code>{locale_source}.lid</code> ссылок
plural	int	Нет	0	Номер множественного индекса в случае множественных строк

Таблица `menu_custom` хранит определения нестандартных меню верхнего уровня (например, первичные ссылки).

Таблица А.58. Таблица `menu_custom` (модуль `menu`)

Имя	Тип	Null	По умолчанию	Описание
<code>menu_name</code>	<code>varchar(32)</code>	Нет		Первичный ключ: уникальный ключ меню. Используется в качестве смещения блока, поэтому длина равна 32, чтобы соответствовать <code>{blocks}.delta</code>
<code>title</code>	<code>varchar(255)</code>	Нет		Заголовок меню, отображаемый в верхней части блока
<code>description</code>	<code>text</code>	Да		Описание меню

Таблица `menu_links` содержит индивидуальные ссылки внутри меню.

Таблица А.59. Таблица `menu_links` (модуль `menu`)

Имя	Тип	Null	По умолчанию	Описание
<code>menu_name</code>	<code>varchar(32)</code>	Нет		Имя меню. Все ссылки с тем же именем меню (таким как <code>navigation</code>) являются частью одного и того же меню
<code>mlid</code>	<code>serial, unsigned autoincrement</code>	Нет		Идентификатор ссылки меню — целочисленный первичный ключ
<code>plid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор родительской ссылки — это <code>mlid</code> ссылки, находящейся выше в иерархии, или 0, если ссылка находится на верхнем уровне меню
<code>link_path</code>	<code>varchar(255)</code>	Нет		Путь Drupal или внешний путь, на который указывает данная ссылка
<code>router_path</code>	<code>varchar(255)</code>	Нет		Для ссылок, соответствующих пути Drupal (0 означает внешний), это значение подключает ссылку к <code>{menu_router}.path</code> для объединений
<code>link_title</code>	<code>varchar(255)</code>	Нет		Текст, отображаемый для ссылки, который может быть модифицирован обратным вызовом заголовка, хранящимся в <code>{menu_router}</code>
<code>options</code>	<code>blob</code>	Да		Сериализованный массив параметров, передаваемых функциям <code>url()</code> или <code>l()</code> , таких как строка запроса или атрибуты HTML
<code>module</code>	<code>varchar(255)</code>	Нет	'system'	Имя модуля, сгенерировавшего эту ссылку
<code>hidden</code>	<code>int:small</code>	Нет	0	Флаг, указывающий на то, должна ли ссылка визуализироваться в меню (1 означает отключенный элемент меню, который может отображаться на административных экранах; -1 — обратный вызов меню; 0 — обычная видимая ссылка)

Окончание табл. А.59

Имя	Тип	Null	По умолчанию	Описание
external	int:small	Нет	0	Флаг-признак указания ссылки на полный URL, начиная с протокола, вроде <code>http://</code> (1 — для внешних, 0 — для внутренних)
has_children	int:small	Нет	0	Флаг-признак того, является ли данная ссылка родительской для других (1 — дочерние ссылки существуют, 0 — нет)
expanded	int:small	Нет	0	Флаг-признак необходимости визуализации данной ссылки в меню в развернутом виде; развернутые ссылки всегда отображают свои дочерние ссылки вместо того, чтобы делать это, только когда ссылка находится в активной цепочке просмотра ссылок (1 — развернута, 0 — нет)
weight	int	Нет	0	Вес ссылки среди ссылок одного и того же уровня того же меню
depth	int:small	Нет	0	Глубина относительно верхнего уровня; ссылка с <code>plid</code> , равным 0, будет иметь глубину 1
customized	int:small	Нет	0	Флаг, указывающий, что пользователь вручную создал или отредактировал ссылку (1 — да, 0 — нет)
p1	int, unsigned	Нет	0	Первый идентификатор <code>mlid</code> в материализованном пути; если $N = \text{depth}$, то значение <code>pN</code> должно быть равно <code>mlid</code> . Если $\text{depth} > 1$, то значение <code>p(N-1)</code> должно быть равно <code>plid</code> . Все значения <code>pX</code> , у которых $X > \text{depth}$, должны быть равны 0. Столбцы <code>p1 ... p9</code> также называются родительскими
p2	int, unsigned	Нет	0	Второй идентификатор <code>mlid</code> в материализованном пути. См. <code>p1</code>
p3	int, unsigned	Нет	0	Третий идентификатор <code>mlid</code> в материализованном пути. См. <code>p1</code>
p4	int, unsigned	Нет	0	Четвертый идентификатор <code>mlid</code> в материализованном пути. См. <code>p1</code>
p5	int, unsigned	Нет	0	Пятый идентификатор <code>mlid</code> в материализованном пути. См. <code>p1</code>
p6	int, unsigned	Нет	0	Шестой идентификатор <code>mlid</code> в материализованном пути. См. <code>p1</code>
p7	int, unsigned	Нет	0	Седьмой идентификатор <code>mlid</code> в материализованном пути. См. <code>p1</code>
p8	int, unsigned	Нет	0	Восьмой идентификатор <code>mlid</code> в материализованном пути. См. <code>p1</code>
p9	int, unsigned	Нет	0	Девятый идентификатор <code>mlid</code> в материализованном пути. См. <code>p1</code>
updated	int:small	Нет	0	Флаг-признак того, что эта ссылка была сгенерирована во время обновления от Drupal 5

522 Приложение А. Справочник по таблицам базы данных

Таблица `menu_router` сопоставляет пути различным обратным вызовам (т.е. обратным вызовам доступа, страницы и заголовка).

Таблица А.60. Таблица `menu_router`

Имя	Тип	Null	По умолчанию	Описание
<code>path</code>	<code>varchar(255)</code>	Нет		Первичный ключ: путь Drupal, описывающий этот элемент
<code>load_functions</code>	<code>blob</code>	Нет		Сериализованный массив имен функций (вроде <code>node_load</code>), вызываемых для загрузки объекта, соответствующего части текущего пути
<code>to_arg_functions</code>	<code>blob</code>	Нет		Сериализованный массив имен функций (вроде <code>user_uid_optional_to_arg</code>), вызываемых для замены части пути маршрутизатора другой строкой
<code>access_callback</code>	<code>varchar(255)</code>	Нет		Обратный вызов, определяющий доступ к пути маршрутизатора; значение по умолчанию — <code>user_access</code>
<code>access_arguments</code>	<code>blob</code>	Да		Сериализованный массив аргументов для обратного вызова доступа
<code>page_callback</code>	<code>varchar(255)</code>	Нет		Имя функции, которая представляет страницу
<code>page_arguments</code>	<code>blob</code>	Да		Сериализованный массив аргументов для обратного вызова страницы
<code>delivery_callback</code>	<code>varchar(255)</code>	Нет	NULL	Имя функции, которая отправляет результат функции <code>page_callback</code> браузеру
<code>fit</code>	<code>int</code>	Нет	0	Числовое представление специфичности пути
<code>number_parts</code>	<code>int:small</code>	Нет	0	Количество частей в этом пути маршрутизатора
<code>context</code>	<code>int</code>	Нет	0	Только для локальных задач (вкладок); контекст локальной задачи, предназначенный для управления его размещением
<code>tab_parent</code>	<code>varchar(255)</code>	Нет		Только для локальных задач (вкладок); путь маршрутизатора родительской страницы (которая также может быть локальной задачей)
<code>tab_root</code>	<code>varchar(255)</code>	Нет		Путь маршрутизатора ближайшей родительской страницы, не являющейся вкладкой. Для страниц, не являющихся локальными задачами, это значение будет совпадать с путем

Окончание табл. А.60

Имя	Тип	Null	По умолчанию	Описание
title	varchar(255)	Нет		Заголовок текущей страницы или заголовок вкладки, если это локальная задача
title_callback	varchar(255)	Нет		Функция, которая изменит заголовок; по умолчанию — <code>t()</code>
title_arguments	varchar(255)	Нет		Сериализованный массив аргументов обратного вызова заголовка. Если этот массив пуст, заголовок будет использован в качестве единственного аргумента обратного вызова заголовка
type	int	Нет	0	Числовое представление типа элемента меню подобного <code>MENU_LOCAL_TASK</code>
theme_callback	varchar(255)	Нет	NULL	Функция, возвращающая имя темы, которая будет использоваться для визуализации данной страницы. Если этот столбец оставлен пустым, будет использоваться тема по умолчанию
theme_arguments	varchar(255)	Нет	NULL	Сериализованный массив аргументов обратного вызова темы
block_callback	varchar(255)	Нет		Имя функции, используемой для визуализации блока на административной странице системы для этого элемента меню
description	text	Нет		Описание этого элемента меню
position	varchar(255)	Нет		Позиция блока (слева или справа) на административной странице системы для данного элемента меню
weight	int	Нет	0	Вес элемента. Элементы с меньшим весом располагаются выше, с большим весом — ниже
include_file	text:medium	Да		Подключаемый для этого элемента файл, в котором обычно находится функция обратного вызова страницы

Таблица `node` является базовой таблицей для нод.

Таблица А.61. Таблица `node` (модуль `node`)

Имя	Тип	Null	По умолчанию	Описание
<code>nid</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный идентификатор ноды
<code>vid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор текущей версии <code>{node_revisions}.vid</code>
<code>type</code>	<code>varchar(32)</code>	Нет		<code>{node_type}.type</code> этой ноды
<code>language</code>	<code>varchar(12)</code>	Нет		<code>{languages}.language</code> этой ноды
<code>title</code>	<code>varchar(255)</code>	Нет		Заголовок этой ноды, всегда трактуемый как простой неразмеченный текст
<code>uid</code>	<code>int</code>	Нет	0	<code>{users}.uid</code> , владеющий этой нодой; изначально — пользователь, создавший ее
<code>status</code>	<code>int</code>	Нет	1	Булевское значение — признак публикации ноды (доступно для просмотра пользователям, не являющимся администраторами)
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix момента создания ноды
<code>changed</code>	<code>int</code>	Нет	0	Временная метка Unix момента последнего сохранения ноды
<code>comment</code>	<code>int</code>	Нет	0	Разрешены ли комментарии для этой ноды: 0 — нет, 1 — комментарии только для чтения; 2 — комментарии для чтения и записи
<code>promote</code>	<code>int</code>	Нет	0	Булевское значение — признак необходимости отображения ноды на титульной странице
<code>sticky</code>	<code>int</code>	Нет	0	Булевское значение, указывающее, должна ли нода отображаться в верхней части списков, в которых она появляется
<code>tnid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор набора переводов для этой ноды, который совпадает с идентификатором ноды исходной статьи каждого набора
<code>translate</code>	<code>int</code>	Нет	0	Булевское значение, указывающее на необходимость обновления данной страницы перевода

Таблица `node_access` идентифицирует пары “область/разрешение”, которые должен предоставить пользователь, чтобы просматривать, обновлять или удалять определенные ноды.

Таблица А.62. Таблица `node_access` (модуль `node`)

Имя	Тип	Null	По умолчанию	Описание
<code>nid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор ноды <code>{node}.nid</code> , затрагиваемой этой записью
<code>gid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор разрешения, которым должен обладать пользователь в указанной области, чтобы получить полномочия этой строки на ноду

Окончание табл. А.62

Имя	Тип	Null	По умолчанию	Описание
realm	varchar(255)	Нет		Область, в которой пользователь должен обладать идентификатором разрешения. Каждая нода доступа к ноте может определять одну или более областей
grant_view	int:tiny, unsigned	Нет	0	Булевское значение, указывающее, может ли пользователей с заданной парой "область/разрешение" просматривать эту ноду
grant_update	int:tiny, unsigned	Нет	0	Булевское значение, указывающее, может ли пользователей с заданной парой "область/разрешение" редактировать эту ноду
grant_delete	int:tiny, unsigned	Нет	0	Булевское значение, указывающее, может ли пользователей с заданной парой "область/разрешение" удалять эту ноду

Таблица `node_comment_statistics` поддерживает статистику нод и комментирующих сообщений для отображения флагов "new" (новый) и "updated" (обновленный).

Таблица А.63. Таблица `node_comment_statistics` (модуль `comment`)

Имя	Тип	Null	По умолчанию	Описание
nid	int, unsigned	Нет	0	Идентификатор ноды <code>{node}.nid</code> , для которой собирается статистика
cid	int	Нет	0	Идентификатор ноды <code>{comment}.nid</code> , для которой собирается статистика
last_comment_timestamp	int	Нет	0	Временная метка Unix последнего комментария внутри данной ноды; получена из <code>{comments}.timestamp</code>
last_comment_name	varchar(60)	Да		Имя последнего автора, отправившего комментарий в данной ноте; получена из <code>{comments}.name</code>
last_comment_uid	int	Нет	0	Идентификатор пользователя последнего автора, отправившего комментарий в данной ноте; из <code>{comments}.uid</code>
comment_count	int, unsigned	Нет	0	Общее количество комментариев в данной ноте

Таблица `node_counter` хранит статистику доступа к нодам.

Таблица А.64. Таблица `node_counter` (модуль `statistics`)

Имя	Тип	Null	По умолчанию	Описание
nid	int	Нет	0	Идентификатор ноды <code>{node}.nid</code> для этой статистики
totalcount	int:big, unsigned	Нет	0	Общее количество просмотров ноды <code>{node}</code>
daycount	int:medium, unsigned	Нет	0	Общее количество просмотров ноды <code>{node}</code> за сегодня
timestamp	int, unsigned	Нет	0	Время последнего просмотра ноды <code>{node}</code>

Таблица `node_revisions` хранит информацию о каждой сохраненной версии ноды.

Таблица A.65. Таблица `node_revisions` (модуль `node`)

Имя	Тип	Null	По умолчанию	Описание
<code>nid</code>	<code>int, unsigned</code>	Нет	0	Нода <code>{node}</code> , к которой относится данная версия
<code>vid</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный идентификатор этой версии
<code>uid</code>	<code>int</code>	Нет	0	Пользователь <code>{user}.uid</code> , создавший эту версию
<code>title</code>	<code>varchar(255)</code>	Нет		Заголовок этой версии
<code>log</code>	<code>text:long</code>	Нет		Запись журнала, описывающая изменения в этой версии
<code>timestamp</code>	<code>int</code>	Нет	0	Временная метка Unix, указывающая время создания этой версии
<code>status</code>	<code>int</code>	Нет	1	Булевское значение, указывающее, опубликована ли нода (на момент создания этой версии); доступно для просмотра пользователями, не являющимися администраторами
<code>comment</code>	<code>int</code>	Нет	0	Разрешены ли комментарии для этой ноды (на момент создания данной версии): 0 — нет, 1 — комментарии закрыты (только для чтения), 2 — комментарии открыты (для чтения и записи)
<code>promote</code>	<code>int</code>	Нет	0	Булевское значение, указывающее, должна ли нода (на момент создания этой версии) отображаться на титульной странице
<code>sticky</code>	<code>int</code>	Нет	0	Булевское значение, указывающее, должна ли нода (на момент создания этой версии) отображаться в верхней части списков, в которых она появляется

Таблица `node_type` хранит информацию обо всех определенных типах ноды `{node}`.

Таблица A.66. Таблица `node_type` (модуль `node`)

Имя	Тип	Null	По умолчанию	Описание
<code>type</code>	<code>varchar(32)</code>	Нет		Машиночитаемое имя этого типа
<code>name</code>	<code>varchar(255)</code>	Нет		Имя типа, читабельное для человека
<code>base</code>	<code>varchar(255)</code>	Нет		Основная строка, используемая для создания обратных вызовов, соответствующих этому типу ноды
<code>description</code>	<code>text:medium</code>	Нет		Краткое описание этого типа ноды
<code>help</code>	<code>text_medium</code>	Нет		Справочная информация, отображаемая пользователю при создании ноды этого типа
<code>has_title</code>	<code>int:tiny, unsigned</code>	Нет		Булевское значение — признак использования типом поля <code>{node}.title</code>
<code>title_label</code>	<code>varchar(255)</code>	Нет		Надпись, отображаемая для поля заголовка в форме редактирования

Окончание табл. А.66

Имя	Тип	Null	По умолчанию	Описание
custom	int:tiny	Нет	0	Булевское значение, указывающее, определен этот тип модулем (0) или пользователем посредством модуля, подобного Content Construction Kit (1)
modified	int:tiny	Нет	0	Булевское значение — признак модификации типа администратором; в настоящее время не используется
locked	int:tiny	Нет	0	Булевское значение, указывающее на возможность изменения администратором машинного имени этого типа
orig_type	varchar(255)	Нет		Исходное машиночитаемое имя этого типа ноды. Оно может отличаться от текущего имени типа, если значение поля locked равно 0

Таблица `openid_association` хранит временную информацию об ассоциации разделяемых ключей для аутентификации OpenID.

Таблица А.67. Таблица `openid_association` (модуль `openid`)

Имя	Тип	Null	По умолчанию	Описание
ipd_endpoint_uri	varchar(255)	Да		URI конечной точки поставщика OpenID
assoc_handle	varchar(255)	Нет		Первичный ключ: используется для ссылки на эту ассоциацию в последующих сообщениях
assoc_type	varchar(32)	Да		Используемый алгоритм подписи: HMAC-HMAC-SHA1 или HMAC-SHA256
session_type	varchar(32)	Да		Допустимые типы ассоциации сеанса: no-encryption, DH-SHA1 и DH-SHA256
mac_key	varchar(255)	Да		Ключ MAC (разделяемый секретный ключ) для этой ассоциации
created	int	Нет	0	Временная метка Unix — момент создания ассоциации
expires_in	int	Нет	0	Время жизни ассоциации в секундах

Таблица `openid_nonce` хранит полученные значения `openid.response_nonce` для каждого URL конечной точки OpenID для предотвращения атак с использованием ретрансляции.

Таблица А.68. Таблица `openid_nonce` (модуль `openid`)

Имя	Тип	Null	По умолчанию	Описание
description	varchar(255)	Да		URI конечной точки поставщика OpenID
nonce	varchar(255)	Да		Значение <code>openid.response_nonce</code>
expires	int	Нет	0	Временная метка Unix, указывающая срок устаревания записи

Таблица `poll` хранит специфическую информацию для нод опросов.

Таблица А.69. Таблица `poll` (модуль `poll`)

Имя	Тип	Null	По умолчанию	Описание
<code>nid</code>	<code>int, unsigned</code>	Нет	0	<code>{node}.nid</code> опроса
<code>runtime</code>	<code>int</code>	Нет	0	Количество секунд после момента <code>{node}.created</code> , в течение которых опрос открыт
<code>active</code>	<code>int, unsigned</code>	Нет	0	Булевское значение — признак открытия опроса

Таблица `poll_choice` хранит информацию обо всех вариантах выбора для всех опросов.

Таблица А.70. Таблица `poll_choice` (модуль `poll`)

Имя	Тип	Null	По умолчанию	Описание
<code>chid</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный ключ: уникальный идентификатор для выбора ответа в опросе
<code>nid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор ноды <code>{node}.nid</code> , к которой относится этот выбор
<code>chtext</code>	<code>varchar(128)</code>	Нет		Текст выбора
<code>chvotes</code>	<code>int</code>	Нет	0	Общее количество голосов за этот выбор, полученных от всех пользователей
<code>weight</code>	<code>int</code>	Нет	0	Порядок сортировки этого выбора среди прочих для той же ноды

Таблица `poll_vote` хранит сведения о голосовании каждого пользователя в каждом опросе.

Таблица А.71. Таблица `poll_vote` (модуль `poll`)

Имя	Тип	Null	По умолчанию	Описание
<code>chid</code>	<code>int, unsigned</code>	Нет	0	Уникальный идентификатор для выбора ответа в опросе
<code>nid</code>	<code>int, unsigned</code>	Нет		Нода <code>{poll}</code> , которой отдан этот голос
<code>uid</code>	<code>int, unsigned</code>	Нет	0	Пользователь <code>{users}.uid</code> , от которого получен данный голос, если только голосование не было анонимным
<code>hostname</code>	<code>varchar(128)</code>	Нет		IP-адрес, с которого поступил голос, если только пользователь не вошел в систему
<code>timestamp</code>	<code>int</code>	Нет		Временная метка Unix момента создания голоса

Таблица `profile_field` хранит информацию о полях профиля.

Таблица A.72. Таблица `profile_field` (модуль `profile`)

Имя	Тип	Null	По умолчанию	Описание
<code>fid</code>	<code>serial, auto increment</code>	Нет		Первичный ключ: уникальный идентификатор поля профиля
<code>title</code>	<code>varchar(255)</code>	Да		Заголовок поля, показанный конечному пользователю
<code>name</code>	<code>varchar(128)</code>	Нет	' '	Внутреннее имя поля, использованное в HTML-форме и в URL
<code>explanation</code>	<code>text</code>	Да		Объяснение поля конечным пользователям
<code>category</code>	<code>varchar(255)</code>	Да		Категория профиля, в которой поле будет сгруппировано с другими
<code>page</code>	<code>varchar(255)</code>	Да		Заголовок страницы, используемый для просмотра по значению поля
<code>type</code>	<code>varchar(128)</code>	Да		Тип поля формы
<code>weight</code>	<code>int</code>	Нет	0	Вес поля относительно других полей профиля
<code>required</code>	<code>int:tiny</code>	Нет	0	Требуется ли от пользователя ввод значения (0 — нет, 1 — да)
<code>register</code>	<code>int:tiny</code>	Нет	0	Видимо ли поле в регистрационной форме пользователя (1 — да, 0 — нет)
<code>visibility</code>	<code>int:tiny</code>	Нет	0	Уровень видимости поля (0 — скрыто, 1 — приватное, 2 — общедоступное на страницах профиля, но не на страницах списка членов, и 3 — общедоступное в профиле и страницах списка)
<code>autocomplete</code>	<code>int:tiny</code>	Нет	0	Включено ли автоматическое завершение ввода (0 — нет, 1 — да)
<code>options</code>	<code>text</code>	Да		Список опций для использования в поле выбора из списка

Таблица `profile_value` хранит значения для полей профиля.

Таблица A.73. Таблица `profile_value` (модуль `profile`)

Имя	Тип	Null	По умолчанию	Описание
<code>fid</code>	<code>int, unsigned</code>	Нет	0	<code>{profile_field}.fid</code> поля
<code>uid</code>	<code>int, unsigned</code>	Нет	0	<code>{users}.uid</code> пользователя профиля
<code>value</code>	<code>text</code>	Да		Значение для поля

Таблица `queue` хранит элементы очередей.

Таблица А.74. Таблица `queue`

Имя	Тип	Null	По умолчанию	Описание
<code>item_id</code>	<code>serial, unsigned autoincrement</code>	Нет		Уникальный идентификатор элемента
<code>name</code>	<code>varchar(255)</code>	Нет		Имя очереди
<code>data</code>	<code>longblob</code>	Да	NULL	Произвольные данные для элемента
<code>expire</code>	<code>int</code>	Нет	0	Временная метка момента истечения срока потребности в элементе
<code>created</code>	<code>int</code>	Нет	0	Временная метка момента создания элемента

Таблица `rdf_mapping` хранит нестандартные RDF-сопоставления для определяемых пользователем типов контента или переопределенные сопоставления, определяемые модулями.

Таблица А.75. Таблица `rdf_mapping`

Имя	Тип	Null	По умолчанию	Описание
<code>type</code>	<code>varchar(128)</code>	Нет		Имя типа объекта, к которому применяется сопоставление (нода, пользователь, комментарий и т.д.)
<code>bundle</code>	<code>varchar(128)</code>	Нет		Имя пакета, к которому применяется сопоставление
<code>mapping</code>	<code>longblob</code>	Да	NULL	Сериализованное сопоставление типа пакета и полей с элементами RDF

Таблица `registry` хранит имя функции, класса или интерфейса и файл, в котором они находятся.

Таблица А.76. Таблица `registry`

Имя	Тип	Null	По умолчанию	Описание
<code>name</code>	<code>varchar(255)</code>	Нет		Имя функции, класса или интерфейса
<code>type</code>	<code>varchar(9)</code>	Нет		Функция, класс или интерфейс
<code>filename</code>	<code>varchar(255)</code>	Нет		Имя файла
<code>module</code>	<code>varchar(255)</code>	Нет		Имя модуля, которому принадлежит файл
<code>weight</code>	<code>int</code>	Нет	0	Порядок, в котором хуки этого модуля должны вызываться относительно других модулей. Модули с одинаковым весом упорядочиваются по имени

Таблица `registry_file` хранит информацию о файлах, которые были проанализированы для создания реестра.

Таблица A.77. Таблица `registry_file`

Имя	Тип	Null	По умолчанию	Описание
<code>filename</code>	<code>varchar(255)</code>	Нет		Путь к файлу
<code>hash</code>	<code>varchar(64)</code>	Нет		Хеш-значение sha-256 содержимого файла, сохраненное при последнем синтаксическом анализе

Таблица `role` хранит роли пользователя.

Таблица A.78. Таблица `role` (модуль `user`)

Имя	Тип	Null	По умолчанию	Описание
<code>rid</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный ключ: уникальный идентификатор роли
<code>name</code>	<code>varchar(64)</code>	Нет		Уникальное имя роли
<code>weight</code>	<code>int</code>	Нет	0	Вес этой роли в списках и пользовательском интерфейсе

Таблица `role_permission` хранит разрешения для пользователей.

Таблица A.79. Таблица `role_permission` (модуль `user`)

Имя	Тип	Null	По умолчанию	Описание
<code>rid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор роли <code>{role}.rid</code> , которой назначены разрешения
<code>permission</code>	<code>varchar(64)</code>	Нет		Отдельное разрешение, которое предоставлено роли, идентифицированной по <code>rid</code>
<code>module</code>	<code>varchar(255)</code>	Нет		Модуль, объявляющий разрешение

Таблица `search_dataset` хранит элементы, подлежащие поиску.

Таблица A.80. Таблица `search_dataset` (модуль `search`)

Имя	Тип	Null	По умолчанию	Описание
<code>sid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор искомого элемента, например, идентификатор ноды
<code>type</code>	<code>varchar(16)</code>	Да		Тип элемента, например — <code>node</code> (нода)
<code>data</code>	<code>text:big</code>	Нет		Список разделенных пробелами слов из элемента
<code>reindex</code>	<code>int, unsigned</code>	Нет	0	Устанавливается для переиндексации ноды

532 Приложение А. Справочник по таблицам базы данных

Таблица `search_index` хранит индекс поиска и ассоциирует слова, элементы и счетчики.

Таблица A.81. Таблица `search_index` (модуль `search`)

Имя	Тип	Null	По умолчанию	Описание
word	varchar(50)	Нет		Слово {search_total}.word, ассоциированный с искомым элементом
sid	int, unsigned	Нет	0	Идентификатор {search_dataset}.sid искомого элемента, к которому относится слово
type	varchar(16)	Да		Тип {search_dataset}.type искомого элемента, к которому относится слово
score	float	Да		Числовая оценка слова. Более высокое значение означает большую важность

Таблица `search_node_links` хранит элементы (наподобие нод), которые связаны с другими нодами; используется для повышения оценок поиска для нод, на которые часто выполняется ссылка.

Таблица A.82. Таблица `search_node_links` (модуль `search`)

Имя	Тип	Null	По умолчанию	Описание
sid	int, unsigned	Нет	0	Идентификатор {search_dataset}.sid искомого элемента, содержащего ссылку на ноду
type	varchar(16)	Нет		Тип {search_dataset}.type искомого элемента, содержащего ссылку на ноду
nid	int, unsigned	Нет	0	Идентификатор {node}.nid, с которым связан данный элемент
caption	text:big	Да		Текст, используемый для ссылки на {node}.nid

Таблица `search_total` хранит итоговые суммы результатов поиска слов.

Таблица A.83. Таблица `search_total` (модуль `search`)

Имя	Тип	Null	По умолчанию	Описание
word	varchar(50)	Нет		Первичный ключ: уникальное слово в поисковом индексе
count	float	Да		Количество вхождений слова в индекс согласно закону Ципфа для уравнивания вероятностного распределения

Таблица `semaphore` хранит семафоры, блокировки и флаги.

Таблица A.84. Таблица `semaphore`

Имя	Тип	Null	По умолчанию	Описание
name	varchar(255)	Нет		Уникальное имя
value	varchar(255)	Нет		Значение семафора
expire	double	Нет		Временная метка Unix, указывающая срок устаревания семафора

Таблица `sequences` хранит идентификаторы.

Таблица A.85. Таблица `sequences`

Имя	Тип	Null	По умолчанию	Описание
<code>value</code>	<code>int</code>	Нет		Значение последовательности

Таблица `sessions` хранит информацию о сеансе пользователя, как анонимного, так и аутентифицированного.

Таблица A.86. Таблица `sessions`

Имя	Тип	Null	По умолчанию	Описание
<code>uid</code>	<code>int, unsigned</code>	Нет		Идентификатор <code>{users}.uid</code> , соответствующий сеансу, или 0 — для анонимного пользователя
<code>sid</code>	<code>varchar(64)</code>	Нет		Первичный ключ: идентификатор сеанса. Значение генерируется Session API в PHP
<code>ssid</code>	<code>varchar(128)</code>	Нет		Идентификатор защищенного сеанса. Значение генерируется Session API в PHP
<code>hostname</code>	<code>varchar(128)</code>	Нет		IP-адрес, последний раз использованный данным идентификатором сеанса (<code>sid</code>)
<code>timestamp</code>	<code>int</code>	Нет	0	Временная метка Unix последнего запроса страницы данным сеансом. Старые записи автоматически удаляются PHP. См. <code>sess_gc()</code>
<code>cache</code>	<code>int</code>	Нет	0	Время последнего сообщения этого пользователя; это значение используется при указании сайтом значения <code>minimum_cache_lifetime</code> . См. <code>cache_get()</code>
<code>session</code>	<code>blob:big</code>	Да		Сериализованное содержимое <code>\$_SESSION</code> , массив пар "имя/значение", который сохраняется между запросами страницы этим идентификатором сеанса. Drupal загружает отсюда значение <code>\$_SESSION</code> в начале каждого запроса и сохраняет его в конце

Таблица `shortcut_set` хранит информацию о наборах ссылок ярлыков.

Таблица A.87. Таблица `shortcut_set`

Имя	Тип	Null	По умолчанию	Описание
<code>set_name</code>	<code>varchar(32)</code>	Нет		Имя <code>{menu_links}.menu_name</code> , под которым сохранены ссылки набора
<code>title</code>	<code>varchar(255)</code>	Нет		Заголовок набора

534 Приложение А. Справочник по таблицам базы данных

Таблица `shortcut_set_users` хранит информацию, которая сопоставляет пользователей с наборами ярлыков.

Таблица A.88. Таблица `shortcut_set_users`

Имя	Тип	Null	По умолчанию	Описание
<code>uid</code>	<code>int</code>	Нет	0	Идентификатор <code>{user}.uid</code> данного набора
<code>set_name</code>	<code>varchar(32)</code>	Нет		Имя <code>{shortcut_set}.set_name</code> , которое будет выведено на экран для этого пользователя

Таблица `simpletest` хранит простые тестовые сообщения.

Таблица A.89. Таблица `simpletest`

Имя	Тип	Null	По умолчанию	Описание
<code>message_id</code>	<code>serial, autoincrement</code>	Нет		Уникальный идентификатор сообщения <code>simpletest</code>
<code>test_id</code>	<code>int</code>	Нет	0	Тестовый идентификатор. О сообщениях, принадлежащих тому же самому идентификатору, сообщается вместе
<code>test_class</code>	<code>varchar(255)</code>	Нет		Имя класса, который создал это сообщение
<code>status</code>	<code>varchar(9)</code>	Нет		Состояние сообщения; ядро понимает состояния передачи, сбоя, исключения.
<code>message</code>	<code>text</code>	Нет		Само сообщение
<code>message_group</code>	<code>varchar(255)</code>	Нет		Группа сообщений, к которой принадлежит это сообщение, например: <code>warning</code> (предупреждение), <code>browser</code> (браузер), <code>user</code> (пользователь)
<code>function</code>	<code>varchar(255)</code>	Нет		Имя функции или метода утверждения, создавшего это сообщение
<code>line</code>	<code>int</code>	Нет	0	Номер строки, в которой вызывается функция
<code>file</code>	<code>varchar(255)</code>	Нет		Имя файла, в котором вызывается функция

Таблица `simpletest_test_id` хранит идентификаторы тестов `simpletest`, служащие для автоматического увеличения значения идентификатора теста, чтобы использовался новый идентификатор.

Таблица A.90. Таблица `simpletest_test_id`

Имя	Тип	Null	По умолчанию	Описание
<code>test_id</code>	<code>serial, autoincrement</code>	Нет		Уникальный идентификатор <code>simpletest</code> , используемый для группирования результатов тестов; при каждом выполнении набора тестов используется новый идентификатор теста
<code>last_prefix</code>	<code>varchar(60)</code>	Да		Последний префикс базы данных, использованный во время тестирования

Таблица `system` содержит список всех модулей, тем и механизмов тем, которые были установлены в файловой системе Drupal.

Таблица A.91. Таблица `system`

Имя	Тип	Null	По умолчанию	Описание
<code>filename</code>	<code>varchar(255)</code>	Нет		Путь к первичному файлу данного элемента относительно корня Drupal; например, <code>modules/node/node.module</code>
<code>name</code>	<code>varchar(255)</code>	Нет		Имя элемента; например, <code>node</code>
<code>type</code>	<code>varchar(12)</code>	Нет		Тип элемента: <code>module</code> , <code>theme</code> или <code>theme_engine</code>
<code>owner</code>	<code>varchar(255)</code>	Нет		“Родитель” темы: может быть либо темой, либо механизмом
<code>status</code>	<code>int</code>	Нет	0	Булевское значение — признак включения данного элемента
<code>bootstrap</code>	<code>int</code>	Нет	0	Булевское значение, указывающее на загрузку этого модуля во время ранней фазы начальной загрузки Drupal (т.е. еще до обращения к кэшу страниц)
<code>schema_version</code>	<code>int:small</code>	Нет	-1	Номер версии схемы базы данных модуля. Если модуль не установлен (т.е. его таблицы не существуют), значение равно -1. Если модуль установлен, значение равно 0 или наибольшему значению N функции <code>hook_update_N()</code> модуля, которая была запущена, либо существовала на момент первоначальной установки модуля
<code>weight</code>	<code>int</code>	Нет	0	Порядок, в котором хуки этого модуля должны вызываться относительно других модулей. Модули с одинаковым весом упорядочиваются по имени
<code>info</code>	<code>blob</code>	Да		Сериализованный массив, содержащий информацию из файла <code>.info</code> модуля; ключи могут выглядеть как <code>name</code> , <code>description</code> , <code>package</code> , <code>version</code> , <code>core</code> , <code>dependencies</code> , <code>dependents</code> и <code>php</code>

Таблица `taxonomy_index` поддерживает денормализованную информацию об отношениях “нода/термин”.

Таблица A.92. Таблица `taxonomy_index`

Имя	Тип	Null	По умолчанию	Описание
<code>nid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор <code>{node}.nid</code> , отслеживаемый этой записью
<code>tid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор термина
<code>sticky</code>	<code>int:tiny</code>	Да	0	Булевское значение, указывающее, является ли нода закрепленной
<code>created</code>	<code>int</code>	Нет	0	Временная метка Unix момента создания ноды

Таблица `taxonomy_term_data` хранит информацию о терминах.

Таблица A.93. Таблица `taxonomy_term_data` (модуль `taxonomy`)

Имя	Тип	Null	По умолчанию	Описание
<code>tid</code>	<code>serial, unsigned autoincrement</code>	Нет		Первичный ключ: уникальный идентификатор термина
<code>vid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор <code>{vocabulary}.vid</code> словаря, в который назначен термин
<code>name</code>	<code>varchar(255)</code>	Нет		Имя термина
<code>description</code>	<code>text:big</code>	Да		Описание термина
<code>format</code>	<code>int:small</code>	Нет	0	Формат <code>{filter_format}.format</code> описания
<code>weight</code>	<code>int</code>	Нет	0	Вес этого термина относительно других терминов

Таблица `taxonomy_term_hierarchy` хранит иерархическое отношение между терминами.

Таблица A.94. Таблица `taxonomy_term_hierarchy` (модуль `taxonomy`)

Имя	Тип	Null	По умолчанию	Описание
<code>tid</code>	<code>int, unsigned</code>	Нет	0	Первичный ключ: идентификатор <code>{term_data}.tid</code> термина
<code>parent</code>	<code>int, unsigned</code>	Нет	0	Первичный ключ: идентификатор <code>{term_data}.tid</code> родителя термина. 0 означает отсутствие родителя

Таблица `taxonomy_vocabulary` хранит информацию словаря.

Таблица A.95. Таблица `taxonomy_vocabulary`

Имя	Тип	Null	По умолчанию	Описание
<code>vid</code>	<code>serial, unsigned autoincrement</code>	Нет		Уникальный идентификатор словаря
<code>name</code>	<code>varchar(255)</code>	Нет		Имя словаря
<code>machine_name</code>	<code>varchar(255)</code>	Нет		Машинное имя словаря
<code>description</code>	<code>text:big</code>	Да	NULL	Описание словаря
<code>hierarchy</code>	<code>int:tiny</code>	Нет	0	Тип иерархии, разрешенной внутри словаря. (0 — отключена, 1 — единственная, 2 — множественная)
<code>module</code>	<code>varchar(255)</code>	Нет		Модуль, создавший этот словарь
<code>weight</code>	<code>int</code>	Нет	0	Вес данного словаря относительно других словарей

Таблица `tracker_node` хранит информацию о времени последнего изменения или комментирования нод.

Таблица A.96. Таблица `tracker_node`

Имя	Тип	Null	По умолчанию	Описание
<code>nid</code>	<code>int</code>	Нет	0	Идентификатор ноды <code>{node}.nid</code> , отслеживаемый этой записью
<code>published</code>	<code>int:tiny</code>	Да	0	Булевское значение, указывающее, опубликована ли нода
<code>changed</code>	<code>int</code>	Нет	0	Временная метка Unix момента последнего сохранения или комментирования ноды

Таблица `tracker_user` хранит информацию о времени последнего изменения или комментирования нод, для каждого пользователя, который создал ноду или один из ее комментариев.

Таблица A.97. Таблица `tracker_user`

Имя	Тип	Null	По умолчанию	Описание
<code>nid</code>	<code>int, unsigned</code>	Нет	0	Идентификатор ноды <code>{node}.nid</code> , отслеживаемый этой записью
<code>uid</code>	<code>int</code>	Нет	0	Идентификатор <code>{users}.uid</code> автора ноды или комментария
<code>published</code>	<code>int:tiny</code>	Да	0	Булевское значение, указывающее, опубликована ли нода
<code>changed</code>	<code>int</code>	Нет	0	Временная метка Unix момента последнего сохранения или комментирования ноды

Таблица `trigger_assignments` сопоставляет триггеры с хуками и назначениями операций модуля триггера.

Таблица A.98. Таблица `trigger_assignments` (модуль `trigger`)

Имя	Тип	Null	По умолчанию	Описание
<code>hook</code>	<code>varchar(32)</code>	Нет		Первичный ключ: имя внутреннего хука Drupal, в котором иницируется действие; например, <code>nodeapi</code>
<code>aid</code>	<code>varchar(255)</code>	Нет		Первичный ключ: <code>{actions}.aid</code> действия
<code>weight</code>	<code>int</code>	Нет	0	Вес назначения триггера относительно других триггеров

Таблица `url_alias` хранит список псевдонимов URL для путей Drupal; пользователь может посетить исходный или целевой путь.

Таблица A.99. Таблица url_alias (модуль path)

Имя	Тип	Null	По умолчанию	Описание
pid	serial, unsigned autoincrement	Нет		Уникальный идентификатор псевдонима пути
source	varchar(255)	Нет		Путь Drupal, для которого определен данный псевдоним; например, node/12
alias	varchar(255)	Нет		Псевдоним пути; например, title-of-the-story
language	varchar(12)	Нет		Язык, для которого предназначен псевдоним; если значение оставлено пустым, псевдоним будет использован для неизвестных языков. Каждый путь Drupal может иметь псевдоним для каждого поддерживаемого языка

Таблица users хранит данные о пользователях.

Таблица A.100. Таблица users (модуль user)

Имя	Тип	Null	По умолчанию	Описание
uid	serial, unsigned	Нет		Первичный ключ: уникальный идентификатор пользователя
name	varchar(60)	Нет		Уникальное имя пользователя
pass	varchar(128)	Нет		Пароль пользователя (хешированный)
mail	varchar(255)	Да		Адрес электронной почты пользователя
theme	varchar(255)	Нет		Тема пользователя по умолчанию
signature	varchar(255)	Нет		Подпись пользователя
signature_ format	int:small	Нет	0	Формат {filter_format}.format подписи
created	int	Нет	0	Временная метка момента создания пользователя
access	int	Нет	0	Временная метка предыдущего обращения пользователя к сайту
login	int	Нет	0	Временная метка последнего входа пользователя в систему
status	int:tiny	Нет	0	Активен пользователь (1) или заблокирован (0)
timezone	varchar(8)	Да		Часовой пояс пользователя
language	varchar(12)	Нет		Язык пользователя по умолчанию
picture	int	Да		Внешний ключ: {file_managed}.fid изображения пользователя
init	varchar(254)	Да		Адрес электронной почты, используемый для начального создания учетной записи
data	longblob	Да		Сериализованный массив пар "имя/значение", которые связаны с пользователем; любые значения формы, отправленные во время редактирования пользователя, сохраняются и загружаются в объект \$user во время выполнения user_load(). Использование этого поля не рекомендуется, и, вероятно, оно исчезнет в следующей версии Drupal

Таблица `users_roles` сопоставляет пользователей ролям.

Таблица А.101. Таблица `users_roles` (модуль `user`)

Имя	Тип	Null	По умолчанию	Описание
<code>uid</code>	<code>int, unsigned</code>	Нет	0	Первичный ключ: идентификатор <code>{users}.uid</code> пользователя
<code>rid</code>	<code>int, unsigned</code>	Нет	0	Первичный ключ: идентификатор <code>{role}.rid</code> роли

Таблица `variable` хранит пары “переменная/значение”, созданные ядром Drupal либо любым другим модулем или темой. Все переменные кэшируются в памяти в начале каждого запроса Drupal, поэтому разработчикам не следует небрежно относиться к сохраненным здесь данным.

Таблица А.102. Таблица `variable`

Имя	Тип	Null	По умолчанию	Описание
<code>name</code>	<code>varchar(128)</code>	Нет		Первичный ключ: имя переменной
<code>value</code>	<code>longblob</code>	Нет		Значение переменной

Таблица `watchdog` содержит журналы всех системных событий.

Таблица А.103. Таблица `watchdog` (модуль `dblog`)

Имя	Тип	Null	По умолчанию	Описание
<code>wid</code>	<code>serial, auto increment</code>	Нет		Первичный ключ: уникальный идентификатор события регистратора (<code>watchdog</code>)
<code>uid</code>	<code>int</code>	Нет	0	Идентификатор <code>{users}.uid</code> пользователя, инициировавшего событие
<code>type</code>	<code>varchar(64)</code>	Нет	''	Тип журнального сообщения, например, <code>user</code> (пользователь) или <code>page not found</code> (страница не найдена)
<code>message</code>	<code>text:big</code>	Нет		Текст журнального сообщения, передаваемого в функцию <code>t()</code>
<code>variables</code>	<code>blob</code>	Нет		Сериализованный массив переменных, который отображает строку сообщения и передается в функцию <code>t()</code>
<code>severity</code>	<code>int:tiny, unsigned</code>	Нет	0	Уровень серьезности события; находится в диапазоне от 0 (авария) до 7 (отладка)
<code>link</code>	<code>varchar(255)</code>	Нет	''	Ссылка для просмотра результатов события
<code>location</code>	<code>text</code>	Нет		URL источника события
<code>referer</code>	<code>varchar(128)</code>	Нет	''	URL ссылающейся страницы
<code>hostname</code>	<code>varchar(128)</code>	Нет	''	Имя хоста пользователя, инициировавшего событие
<code>timestamp</code>	<code>int</code>	Нет	0	Временная метка Unix момента возникновения события

ПРИЛОЖЕНИЕ Б

Ресурсы

Для разработчиков Drupal доступны многие ресурсы. Эти ресурсы могут помочь в изучении Drupal, демонстрируя возможности системы, а также натолкнуть на новые идеи и решения. Хорошей отправной точкой могут быть веб-сайт Drupal.org (<http://drupal.org>), веб-сайт групп Drupal (<http://groups.drupal.org>), местная группа пользователей Drupal (<http://drupal.org/event>), полугодовые собрания DrupalCon, региональные учебные лагеря Drupal, код-спринты, активный чат (<http://drupal.org/irc>), списки рассылки (<http://drupal.org/mailling-lists>), планета Drupal (<http://drupal.org/planet>) и глобальное сообщество, которое всегда направляет разработчика на форумы Drupal.org (<http://drupal.org/forum>). Однако этот список далеко не исчерпывающий, и в настоящем приложении приводится перечень других полезных ресурсов.

Код

Ниже перечислены ресурсы, содержащие код Drupal.

Drupal CVS

Исходный код Drupal, включая ядро и дополнительные модули, хранится в системе управления исходным кодом. Эта система позволяет множеству разработчиков совместно работать над одним проектом (например, ядром Drupal). При этом система дает возможность каждому участнику извлечь код для работы над ним, на это время предотвращает изменения взятого кода другими участниками и затем позволяет участнику вернуть измененный код обратно, делая его доступным всем остальным. Исторически сложилось, что данное хранилище было основано на CVS-системе, однако в конце 2010 г. проект Drupal переместился на Git (<http://git-scm.com>) — другую мощную распределенную систему управления исходным кодом.

Загрузить и поучаствовать в написании исходного кода можно по адресу <http://drupal.org/download>.

Примеры

Одним из наилучших ресурсов, доступных для разработчиков, является полный набор примеров кода, который можно загрузить из <http://drupal.org/project/examples>. В набор входят десятки примеров программ, раскрывающих практически каждый аспект Drupal. Код хорошо документирован и предоставляется членами сообщества Drupal. Сопровождением проекта занимается Ренди Фей (Randy Fay), который всегда рад ответить на вопросы и предложения по поводу примеров.

Справочная информация по Drupal API

Создание кода Drupal, как было показано в этой книге, полагается на использование API-интерфейсов Drupal. Полный список API-интерфейсов доступен по адресу <http://api.drupal.org>. Щелкните на вкладке для версии Drupal 7, чтобы просмотреть только API-интерфейсы Drupal 7.

Рекомендации по безопасности

Рекомендации по безопасности доступны по электронной почте или с помощью RSS-ленты на странице <http://drupal.org/security>. Зайдя на сайт <http://drupal.org>, на этой странице можно подписаться на получение рекомендаций. Если вы планируете заняться написанием защищенного кода, перечитайте главу 21 и посетите страницу <http://drupal.org/writing-secure-code>.

Обнаружив проблемы с безопасностью Drupal, зайдите на страницу <http://drupal.org/security-team> и прочитайте раздел о том, как сообщить о проблеме.

Обновление модулей

Когда API-интерфейс изменяется в связи с появлением нового выпуска Drupal, формальные результаты изменения документируются на странице <http://drupal.org/update/modules>. Эта страница незаменима для поддержания модулей в синхронизированном состоянии с модификациями, внесенными в кодовую базу Drupal.

Обновление тем

Обновления тем могут быть найдены по адресу <http://drupal.org/update/theme>. На этой странице вы найдете новые версии тем, а также новые добавленные темы.

Руководства

Онлайновые руководства, размещенные на странице <http://drupal.org/handbooks>, постоянно обновляются и улучшаются. Сюда также отправляются документы типа HOWTO (практические рекомендации), содержащие пошаговые инструкции.

Форумы

Форумы на странице <http://drupal.org/forum> — замечательное место, где можно получить помощь по Drupal. Часто кто-то уже сталкивался с проблемой, которая возникла у вас, и документировал свой опыт на форумах. Что касается проблем, вызванных явными ошибками в дополнительных модулях, однако лучше внести запись в очередь сообщений об ошибках модуля, поскольку разработчики скорее увидят его там, чем в форумах.

Совет. Попробуйте воспользоваться поисковым механизмом, чтобы ограничить количество результатов, получаемых с <http://drupal.org>. Например, запрос "installation profiles" site:drupal.org в Google обеспечит поиск на сайте <http://drupal.org> строки "installation profiles".

Списки рассылки

Поддерживается множество списков рассылки на разнообразные темы. Управление подпиской удобно производить на странице <http://lists.drupal.org/mailman/listinfo>. Полный перечень списков рассылки доступен по адресу <http://lists.drupal.org>. Ниже перечислены наиболее интересные списки рассылки.

Список рассылки **development** (разработка)

Этот список рассылки предназначен для разработчиков Drupal и включает общие дискуссии о направлении развития системы Drupal, обсуждение связанных с ее разработкой вопросов и преимуществ различных подходов. Если делается существенное изменение, оно обычно обсуждается здесь.

Список рассылки **themes** (темы)

Для разработчиков тем; обсуждаются проблемы, связанные с темизацией Drupal.

Список рассылки **translations** (переводы)

Список рассылки для тех, кто переводит интерфейс Drupal на другие языки.

Группы пользователей и группы по интересам

Местные или региональные группы пользователей и те, которые работают над отдельными аспектами Drupal, могут использовать инфраструктуру сайта <http://groups.drupal.org> для организации и взаимодействия. Особенно интересной для начинающих разработчиков может стать группа Drupal Dojo (<http://groups.drupal.org/drupal-dojo>), цель которой — повысить квалификацию начинающих разработчиков.

Интернет-чат

Интернет-чат (Internet Relay Chat — IRC) жизненно важен для взаимодействия сообщества. Это не только отличный способ получения быстрой и эффективной поддержки, но также путь к изучению нового материала и вовлечению в сообщество. Сообщество Drupal использует сеть FreeNode IRC (irc.freenode.net). Если вам необходим немедленный ответ или просто нужно посмотреть, о чем разговаривают разработчики, воспользуйтесь чатом. Для доступа к irc.freenode.net понадобится клиент IRC.

Три перечисленных ниже канала будут интересны большинству разработчиков.

- **#drupal**. Это своего рода “гостиная” для обсуждения разнообразных вопросов. Здесь говорят об исправлениях, которые необходимо пересмотреть, отвечают на короткие вопросы о том, как сделать то-то и то-то, что другие недавно сделали на Drupal и т.п. Если проблема связана с Drupal и не требует углубленной проработки или повышенного внимания, этот канал — как раз для ее обсуждения.
- **#drupal-contribute**. Это место для общих вопросов по ядру и дополнительным модулям, продвижению, пропаганде, инфраструктуре и сообществу Drupal. Основное правило: если, в конечном счете, проблема выходит на инфраструктуру Drupal.org, она должна обсуждаться на этой канале.
- **#drupal-support**. Это место для длительных или углубленных обсуждений, связанных с поддержкой. В то время как краткие вопросы нормальны для канала **#drupal**, некоторые дискуссии слишком длинны или требуют спокойного обсуждения — для них и предназначен канал **#drupal-support**.

Интерес могут вызвать и другие тематические каналы.

- `#drupal-themes`. Предназначен для углубленных или длительных обсуждений вопросов, связанных с поддержкой тем.
- `#drupal-accessibility`. Предназначен для обсуждения вопросов, связанных с кодированием и доступностью, а также с расширениями Drupal.
- `#drupal-consultants`. Предназначен для консультантов по Drupal. Здесь можно получить платную помощь, а также рекомендации по лучшим методикам и бизнес-стороне Drupal. Задавайте здесь вопросы, только если готовы оплатить за полученные сведения.
- `#drupal-design`. Предназначен для дизайнеров. Обсуждаются вопросы, связанные с чистым дизайном, а не с темами.
- `#drupal-docs`. Предназначен для команды подготовки документации. Обсуждаются вопросы по написанию и работе с руководствами на Drupal.org.
- `#drupal-dojo`. Предназначен для группы Drupal Dojo (<http://groups.drupal.org/drupaldojo>).
- `#drupal-ecommerce`. Предназначен для группы электронной коммерции (E-commerce).
- `#drupal-elearning`. Предназначен для обсуждения модулей электронной обучения и применения Drupal в этом процессе.
- `#drupal-facebook`. Предназначен для группы Facebook API. Здесь обсуждаются все вопросы, связанные с интеграцией Drupal и Facebook, как через дополнительные модули, так и посредством API-интерфейса.
- `#drupal-fit`. Предназначен для обсуждения фитнес-мероприятий на встречах, в учебных лагерях и конференциях Drupal.
- `#drupal-geo`. Предназначен для тех, кто интересуется темами картографии, расположения и географии.
- `#drupal-groups`. Предназначен для организаторов групп. Позволяет обмениваться советами по созданию местных групп пользователей Drupal.
- `#drupal-html5`. Предназначен для обсуждения реализации html5 в Drupal.
- `#drupal-ngo`. Предназначен для открытых дискуссий о наиболее эффективном использовании Drupal для неприбыльных и неправительственных организаций.
- `#drupal-seo`. Предназначен для обсуждения вопросов, связанных с поисковой оптимизацией и разработкой модулей.
- `#drupal-ubercart`. Предназначен для обсуждения вопросов, связанных с поддержкой и разработкой Ubercart.
- `#drupal-usability`. Предназначен для команды, занимающейся оценкой удобства использования (Usability). Обсуждаются изменения в пользовательском интерфейсе и общие проблемы удобства.
- `#drupal-vcs`. Предназначен для обсуждения вопросов, связанных с системами управления версиями.
- `#drupal-l10n`. Предназначен для переводчиков Drupal, менеджеров групп и пользователей localize.drupal.org.
- `#open_atrium`. Предназначен для обсуждения вопросов, связанных с Open Atrium — профиле инсталляции для управления проектами, основанными на Drupal.

Существует также и ряд региональных каналов.

Северная Америка

- #drupal-alaska для группы из Аляски.
- #drupal-bayarea для группы из района залива Сан-Франциско.
- #drupal-boston для групп Drupal из района Бостона.
- #cdmug для группы из Чикаго (www.cdmug.org).
- #drupal-colorado для пользователей Drupal из Колорадо.
- #drupal-idaho для пользователей Drupal из Айдахо.
- #drupal-florida для пользователей Drupal из Флориды.
- #drupal-georgia для пользователей Drupal из Джорджии.
- #drupal-la для группы Drupal из Лос-Анджелеса.
- #drupal-nc для пользователей Drupal из Северной Каролины.
- #drupal-nebraska для пользователей Drupal из Небраски.
- #drupal-nj для группы Drupal из Нью-Джерси.
- #drupal-nyc для группы Drupal из Нью-Йорка.
- #drupaldelphia для группы пользователей Drupal из Филадельфии.
- #drupal-pdx для группы пользователей Drupal из района Портленда, шт. Орегон.
- #drupal-seattle для группы пользователей Drupal из района Сиэтла.
- #drupal-dugto для группы пользователей Drupal из Торонто.
- #drupal-vancouver для группы пользователей Drupal из района Ванкувера.
- #drupal-pnw для группы пользователей Drupal из района Тихоокеанского Северо-Запада.

Европа

- #drupal.cat для пользователей Drupal из Каталонии.
- #drupal.de для пользователей Drupal из Германии.
- #drupal-denmark для группы пользователей Drupal из Дании (<http://drupaldanmark.dk/>).
- #drupal-fr для франкоязычных пользователей Drupal.
- #drupal-el для пользователей Drupal из Греции.
- #drupal-es для испаноязычных пользователей Drupal (<http://groups.drupal.org/spanish>).
- #drupal.hu для пользователей Drupal из Венгрии.
- #drupal-italia для группы пользователей Drupal из Италии (www.drupalitalia.org).
- #drupal-nl для пользователей Drupal из Голландии и Бельгии.
- #drupal-norge для пользователей Drupal из Норвегии.
- #drupal-pl для пользователей Drupal из Польши (<http://groups.drupal.org/poland>).

- #drupal-pt для пользователей Drupal из Португалии (<http://groups.drupal.org/portugal>).
- #drupal-ro для пользователей Drupal из Румынии.
- #drupal-ru для пользователей Drupal из России.
- #drupal-sr для пользователей Drupal Сербии.
- #drupal-se для пользователей Drupal из Швеции.
- #drupal-tr для пользователей Drupal из Турции.
- #drupaluk для разработчиков Drupal из Великобритании.
- #drupal-sl для пользователей Drupal из Словении.

Азия

- #drupal-in для сообщества Drupal из Индии.
- #drupal-israel для сообщества Drupal из Израиля (www.drupal.org.il).
- #drupal-jp для сообщества Drupal из Японии.
- #drupal-sg для пользователей Drupal из Сингапура.
- #drupal-china для пользователей Drupal из Китая.
- #drupal-tr для пользователей Drupal из Турции.
- #drupal-fa для пользователей Drupal из Персидского залива.

Латинская Америка / Карибский бассейн

- #drupal-br для группы пользователей Drupal из Бразилии.
- #drupal-peru для пользователей Drupal из Перу (<http://groups.drupal.org/peru>).

Океания

- #drupal-au для разработчиков и пользователей Drupal из Австралии.
- #drupal-nz для пользователей Drupal из Новой Зеландии.

Африка

- #drupal-mu для пользователей Drupal из острова Маврикий.
- #drupal-za для пользователей Drupal из Южной Африки.

Если вы являетесь новичком в IRC и желаете принимать участие в сообществе IRC, чат FreeNode IRC доступен по адресу <http://webchat.freenode.net>. Дополнительные сведения о подключении к IRC можно найти в статье, посвященной выбору подходящего клиента и подключению к IRC, по адресу <http://groups.drupal.org/node/2326>. Можно также просмотреть видеоролик, посвященный подключению к IRC (<http://drupal.org/node/424300>).

Существует несколько простых команд IRC, которые позволяют более эффективно работать с каналами IRC.

После входа в IRC имеет смысл зарегистрировать свое имя с помощью NickServ, чтобы никто другой не смог выступать под вашим псевдонимом. О регистрации псевдонима читайте <http://freenode.net/faq.shtml#nicksetup>.

Из уважения к Drupalicon и другим членам чата не включайте слово “Drupal” или любую вариацию с ним в свой псевдоним.

Большинство применяет в качестве псевдонима в IRC свои имена пользователей Drupal. Если желаемое имя уже выдано FreeNode, дополните его дефисом или числовой комбинацией. Многие клиенты IRC автоматически добавляют знак подчеркивания к псевдониму, если он уже занят. Хорошо узнаваемый псевдоним позволит согласованно работать с IRC и сайтом Drupal.

В случае применения разных имен в IRC и сайте Drupal.org можно воспользоваться маской (cloak) IRC. Информация по ее получению доступна по адресу <http://groups.drupal.org/node/5403>.

Видеоролики

Иногда некоторые концепции трудно описать, но легко продемонстрировать. Постоянно растущая коллекция видеороликов и экранных снимков доступна по адресу <http://drupal.org/videocasts>.

Блоги

Блоги — это сетевые дневники. Многие разработчики имеют блоги, в которых делятся собственным опытом использования Drupal.

Конференции

Сообщество Drupal периодически собирается на конференциях, где проводятся презентации новых средств, дискуссии и масса развлечений. Обычно конференция проходит весной в Северной Америке и осенью в Европе. Конференции — замечательный способ узнать о Drupal, установить связи и завести новых друзей. Если у вас есть возможность поехать на такую конференцию, обязательно воспользуйтесь ею. Подробности можно найти по адресу <http://drupalcon.org>. IRC-канал #drupalcon используется перед и во время конференций для регистрации новых участников.

Часто перед или после конференции Drupal планируется код-спринт.

Спонсорство

Спонсоры — самый ценный актив Drupal и причина того, что Drupal продолжает развиваться не только как платформа, но также и как сообщество.

На странице <http://drupal.org/contribute> вы можете внести свой вклад в Drupal не только через участие в разработке, но также и через подготовку документации, переводы на другие языки, повышение удобства и простоты использования, пожертвования, маркетинг и т.п. Эта страница поможет принять участие в спонсорстве Drupal на любом уровне.

Предметный указатель

A

Apache, 443
настройка, 441
API (Application Programming Interface), 217

B

Boost, 446

C

ССК (Content Construction Kit), 155

D

Drupal, 23
оптимизация, 436
события Drupal, 48
ядро, 24

F

Form API, 247

I

ISAPI (Internet Server Application
Programming Interface), 30

J

jQuery, 344; 357

L

Linux, 447

N

Nginx, 443

P

PDO (PHP Data Object), 95
PHP
оптимизация, 438
Pressflow, 444

V

Varnish, 444

X

XML-RPC, 393

Б

База данных
определение параметров, 94
оптимизация, 450
подключение
к базе данных, 95
к нескольким базам данных, 113
репликация, 456
секционирование, 456
уровень абстракции, 95
Балансировка загрузки, 455
Безопасность
AJAX, 421
API-интерфейса форм, 422
учетной записи суперпользователя, 422
Блок (block), 28; 204
определение, 207
размещение, 207
создание, 209
специальный, 205
стандартный, 204

Д

Действие (action), 50
идентификаторы действий, 62
расширенное (advanced), 55
хранение действий, 62
Диспетчеризация (dispatching), 68
меню, 70

З

Запрос, 101
простой, 97

И

Идентификатор
ноды (node ID), 134
разрешения (grant ID), 152
Индексатор, 280
Инсталляционный профиль, 462
Интерфейс
Form API, 217
свойства, 247
администрирования, 25

К

Клиент, 394
Кэширование, 323; 436
 блоков, 328
 отключение кэширования, 327
 статической страницы, 328
 страниц, 328
 фильтрованного текста, 326

Л

Локализация, 367

М

Маршрутизация (routing), См.
 Диспетчеризация (dispatching), 68
Массив, 427
Меню
 вложенное, 77
 флаги типов для элементов меню, 88
Модуль, 25; 32
 Boost, 446

Н

Нода (node), 27
 идентификатор ноды (node ID), 134
 модуль ноды, 137

О

Объект
 \$user, 116
Оптимизация PHP, 438

П

Поиск
 HTML-индексатор поиска, 280
 временного каталога, 301
 файлов в каталоге, 301
 форма поиска по умолчанию, 276
Поле, 27; 155
 создание произвольного поля, 160
 типы полей в ядре Drupal 7, 161
 форматер поля, 163
Пользователи, 116
 регистрация пользователя, 122
Протокол
 XML-RPC, 393
Профиль
 инсталляционный, 462
Путь (path), 68

С

Сеанс, 335
 жизненный цикл сеанса, 340
Сервер XML-RPC, 398
Словарь
 иерархический, 308
 на основе модулей, 314
 одноуровневый, 308
Событие (event), 48; 50
Строка, 367
 замена строк с помощью модуля
 локализации, 371
 конкатенация строк, 428
 перевод на определенный язык, 379

Т

Таблица
 accesslog, 497
 actions, 498
 aggregator_category, 498
 aggregator_category_feed, 498
 aggregator_category_item, 499
 aggregator_feed, 499
 aggregator_item, 500
 authmap, 500
 batch, 500
 block_custom, 502
 blocked_ips, 502
 blocks, 501
 blocks_node_type, 502
 blocks_role, 502
 book, 503
 cache, 325; 503
 cache_block, 503
 cache_bootstrap, 504
 cache_field, 504
 cache_filter, 504
 cache_form, 505
 cache_image, 505
 cache_menu, 505
 cache_page, 506
 cache_path, 506
 cache_update, 506
 comment, 507
 contact, 508
 date_format_locale, 508
 date_formats, 508
 date_format_type, 508
 field_config, 509
 field_config_entity_type, 509

- field_config_instance, 510
- field_data_body, 510
- field_data_comment_body, 511
- field_data_field_image, 511
- field_data_field_tags, 512
- field_data_taxonomy_forums, 512
- field_revision_body, 513
- field_revision_comment_body, 513
- field_revision_field_image, 514
- field_revision_field_tags, 514
- field_revision_taxonomy_forums, 515
- file_managed, 294; 515
- file_usage, 516
- filter, 516
- filter_format, 516
- flood, 517
- forum, 517
- forum_index, 517
- history, 518
- image_effects, 518
- image_styles, 518
- languages, 518
- locales_source, 519
- locales_target, 519
- menu_custom, 520
- menu_links, 520
- menu_router, 522
- node, 524
- node_access, 524
- node_comment_statistics, 525
- node_counter, 525
- node_revisions, 526
- node_type, 526
- openid_association, 527
- openid_nonce, 527
- poll, 528
- poll_choice, 528
- poll_vote, 528
- profile_field, 529
- profile_value, 529
- queue, 530
- rdf_mapping, 530
- registry, 530
- registry_file, 531
- role, 531
- role_permission, 531
- search_dataset, 531
- search_index, 532
- search_node_links, 532
- search_total, 532
- semaphore, 532
- sequences, 533
- sessions, 339; 533
- shortcut_set, 533
- shortcut_set_users, 534
- simpletest, 534
- simpletest_test_id, 534
- system, 535
- taxonomy_index, 535
- taxonomy_term_data, 536
- taxonomy_term_hierarchy, 536
- taxonomy_vocabulary, 536
- tracker_node, 537
- tracker_user, 537
- trigger_assignments, 537
- upload, 294
- url_alias, 538
- users, 538
- users_roles, 539
- variable, 539
- watchdog, 539
- временная, 114
- обслуживание таблиц, 110
- оптимизация таблиц, 460
- создание таблиц, 103
- удаление таблиц при деинсталляции, 111
- Таксономия (taxonomy), 306
- Теги (tag), 112
- Текст
 - HTML, 406
 - простой, 406
 - форматированный, 406
- Тема, 27; 173
 - добавление областей в тему, 180
 - создание темы, 174
 - установка готовой темы, 173
- Темизация (theming), 31
- Тестирование, 480
- Тип поля
 - blob, 109
 - Boolean, 161
 - datetime, 108
 - Decimal, 161
 - File, 161
 - float, 108; 161
 - Image, 161
 - int, 107; 161
 - List, 161
 - Long text, 161
 - Long text and Summary, 161
 - numeric, 108
 - serial, 108

Term Reference, 161
 text, 107; 161
 varchar, 106
 char, 107
 приведение типов, 425
 Триггер (trigger), 50

Ф

Файл

block.tpl.php, 195
 bootstrap.inc, 338
 enhanced.info, 463
 enhanced.install, 464
 enhanced.profile, 464
 field.tpl.php, 194
 .htaccess, 337
 html.tpl.php, 186
 node.tpl.php, 191
 page.tpl.php, 187; 190
 .po, 380
 .pot, 379
 region.tpl.php, 191
 settings.php, 337; 369
 безопасность файлов, 416
 общедоступный, 290
 права доступа к файлу, 416
 приватный, 291
 шаблонный из ядра Drupal, 185

Фильтры, 265

инсталляция фильтра, 268
 создание специального фильтра, 270

Форма

визуализация формы, 222
 динамическая, 240
 идентификаторы формы, 227
 изменение, 239
 обработка, 217
 отправка, 224
 программная, 240
 перестройка, 237
 поиска по умолчанию, 276
 построение, 221
 проверка достоверности, 223
 свойства формы, 226
 создание базовых форм, 224
 темизация форм, 230
 элементы формы, 251

Формат

текстовый, 266

Функция

actions_do(), 62
 cache_clear_all(), 333
 cache_get(), 332
 cache_get_multiple(), 332
 cache_is_empty(), 332
 cache_set(), 331
 check_plain(), 407
 creativejuice_filter_process(), 271
 creativejuice_filter_tips(), 273
 db_query(), 97; 412
 drupal_add_js(), 347
 drupal_alter(), 59
 drupal_form_submit(), 240
 drupal_write_record(), 102
 file_create_url(), 300
 filter_xss(), 409
 filter_xss_admin(), 411
 form_set_error(), 235
 form_set_value(), 235
 hook_access(), 142
 hook_alter_query(), 414
 hook_delete(), 146
 hook_field_formatter_info, 163
 hook_field_schema(), 162
 hook_field_validate(), 162
 hook_filter_info(), 271
 hook_flush_caches(), 334
 hook_form_alter(), 239
 hook_forms(), 233
 hook_insert(), 145
 hook_load(), 146
 hook_menu(), 72
 hook_node_update_index(), 283
 hook_permission(), 142
 hook_query_alter(), 112
 hook_schema_alter(), 111
 hook_search_page(), 277
 hook_trigger_info(), 63
 hook_update(), 146
 hook_update_index(), 284
 hook_user_view(), 120
 hook_validate(), 144
 hook_view(), 147
 hook_widget_form(), 165
 hook_xmlrpc(), 400
 hook_xmlrpc(), 399
 taxonomy_get_children(), 320
 taxonomy_get_parents(), 320

taxonomy_get_parents_all(), 320
 taxonomy_get_term_by_name(), 319
 taxonomy_get_tree(), 320
 taxonomy_get_vocabularies(), 317
 taxonomy_load_term(), 318
 taxonomy_select_nodes(), 317
 taxonomy_term_delete(), 320
 taxonomy_term_save(), 319
 taxonomy_vocabulary_delete(), 318
 taxonomy_vocabulary_load(), 317
 taxonomy_vocabulary_save(), 317
 theme(), 198
 to_arg(), 85
 variable_get(), 47
 тестирования, 491

Х

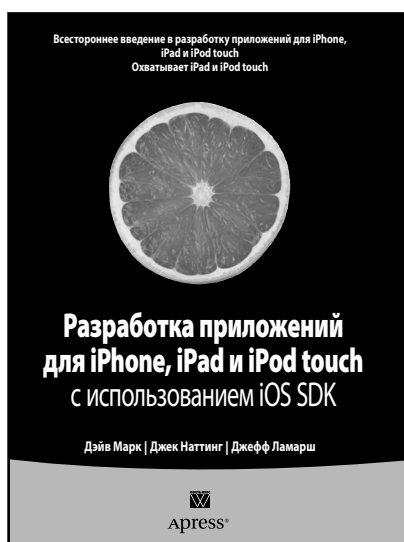
Хуки (hook), 25; 34; 48
 аутентификации, 304
 блочные, 209
 пользователя, 119
 реализация хука, 34

Я

Ядро Drupal, 24
 Язык
 поддержка
 многоязыковая, 387
 с письмом справа налево, 381
 согласование языков, 382
 установка перевода на определенный
 язык, 379

РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ iPhone, iPad И iPod touch С ИСПОЛЬЗОВАНИЕМ iOS SDK

**Дэйв Марк
Джек Наттинг
Джефф Ламарш**



www.williamspublishing.com

Книга представляет собой руководство, в котором подробно рассмотрены вопросы разработки приложений для устройств iPhone, iPad и iPod touch с использованием самых новых версий комплекта iOS SDK и системы iOS. Все проекты были созданы с помощью комплекта инструментов Xcode 3.2.5. Цель книги — объяснить, как работают приложения для системы iOS и продемонстрировать процесс их создания шаг за шагом, ничего не пропуская. Проработав книгу, читатели смогут самостоятельно разрабатывать профессиональные приложения для устройств iPhone, iPad и iPod touch, использующие новейшие возможности системы iOS. Книга предназначена для программистов любого уровня.

ISBN 978-5-8459-1725-6 в продаже

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ С ПОМОЩЬЮ PHP И MySQL

БИБЛИОТЕКА РАЗРАБОТЧИКА

ЧЕТВЕРТОЕ ИЗДАНИЕ

**Люк Веллинг
Лора Томсон**



www.williamspublishing.com

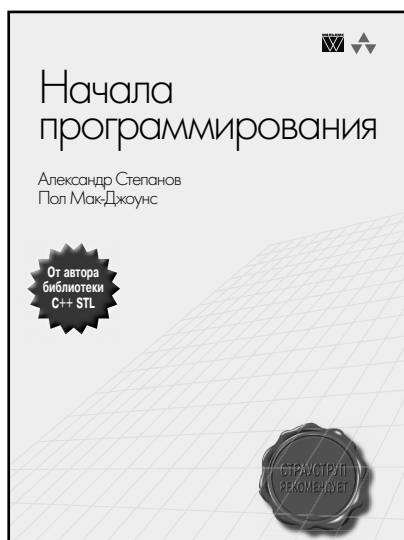
Эта книга предназначена для тех, кто знаком с основами HTML и ранее разрабатывал программы на современных языках программирования, но, возможно, не занимался программированием для веб или не использовал реляционные базы данных. В ней подробно описано применение последних версий PHP и MySQL для построения крупных коммерческих веб-сайтов. Основное внимание в книге уделено реальным приложениям. Здесь рассматриваются как простые интерактивные системы приема заказов, так и различные аспекты электронных систем продажи и безопасности во взаимосвязи с созданием реального веб-сайта. Подробно описаны все стадии разработки типовых проектов на PHP и MySQL, в числе которых служба веб-почты, приложение поддержки веб-форумов и электронный книжный магазин. Книга ориентирована на профессиональных разработчиков, но будет полезной и начинающим.

ISBN 978-5-8459-1574-0

в продаже

НАЧАЛА ПРОГРАММИРОВАНИЯ

**Александр Степанов,
Пол Мак-Джоунс**



www.williamspublishing.com

В настоящей книге применяется дедуктивный подход к программированию, основанный на объединении программ с абстрактными математическими теориями, которые обеспечивают их работу. Представлены вместе описания этих теорий, алгоритмы, записанные с точки зрения этих теорий, а также теоремы и леммы, описывающие их свойства. Реализация алгоритмов на реальном языке программирования является центральной темой книги. Эта книга предназначена для тех, кто стремится глубже понять суть программирования, будь то профессиональные программисты или ученые и инженеры, для которых программирование составляет важную часть их профессиональной деятельности.

ISBN 978-5-8459-1708-9 **в продаже**

PHP И jQuery для ПРОФЕССИОНАЛОВ

Джейсон Ленгсторф



www.williamspublishing.com

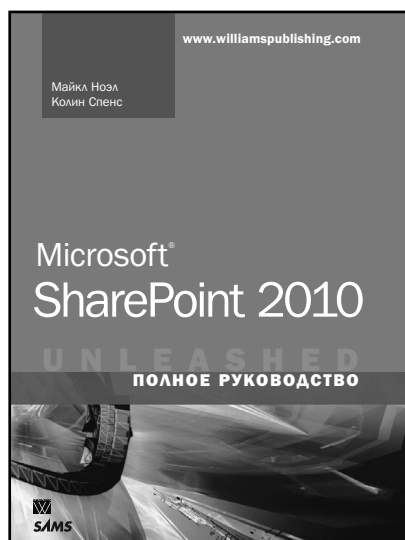
В этой книге вы найдете все необходимое для того, чтобы приступить к разработке мощных веб-приложений на основе jQuery, AJAX и объектно-ориентированных средств PHP. Следуя приведенным в книге рекомендациям, вы в короткие сроки научитесь применять передовые методы разработки PHP-приложений, сочетая их с инструментами jQuery для создания пользовательских интерфейсов с высокой степенью интерактивности.

В процессе работы над примером приложения, играющего роль центрального проекта в данной книге, вы изучите основы объектно-ориентированного программирования на PHP и приобретете достаточные навыки работы с библиотекой jQuery, даже если вы абсолютный новичок в этой области. В книге дается множество практических советов и описан ряд профессиональных приемов, вооружившись которыми вы сможете сразу же приступить к созданию привлекательных и дружелюбных по отношению к пользователю приложений, в полной мере использующих всю мощь AJAX.

ISBN 978-5-8459-1693-8 в продаже

MICROSOFT® SHAREPOINT 2010 ПОЛНОЕ РУКОВОДСТВО

**Майкл Ноэл
Колин Спенс**



www.williamspublishing.com

ISBN 978-5-8459-1728-7

Книга известных профессионалов в области информационных технологий посвящена новой версии популярной в настоящее время системы управления документооборотом для широкого круга пользователей — SharePoint 2010. Подробно рассматриваются такие вопросы, как планирование и развертывание инфраструктуры SharePoint, интеграция с существующими офисными пакетами, разработка страниц, рабочих пространств, библиотек, сайтов и порталов SharePoint, организация эффективного поиска информации, мониторинг, обслуживание и администрирование существующей среды, а также многое другое. Большое внимание уделено интегрированию содержимого с помощью веб-приложений Office и служб Excel Service, Access Service и Visio Graphics Service, разработке собственных решений с помощью SharePoint Designer 2010 и Visual Studio 2010, построению решений бизнес-аналитики с помощью PerformancePoint и службы Business Connectivity Services.

в продаже

РАСКРУТКА ВЕБ-САЙТА

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

Н.В. Евдокимов
И.В. Лебединский



www.williamspublishing.com

Эта книга — совершенно уникальное явление в литературе по продвижению веб-сайтов. Нигде больше вы не найдете столь полного и в то же время простого и понятного изложения как теоретических концепций, так и сугубо практических методов и рекомендаций. Процедура продвижения сайта раскрыта в этом издании во всей своей полноте — от планирования самого сайта, до мероприятий по контролю и корректировке достигнутых результатов. Самые современные понятия, технологии и подходы обсуждаются в строгой последовательности и предлагаются читателю к ежедневному применению в его практической деятельности. Твердая теоретическая база, подкрепленная “живыми”, взятыми из реальной жизни примерами, четкая структура, живой и образный язык — все это делает данное руководство настольной книгой как для начинающих, так и для опытных специалистов в области интернет-маркетинга.

ISBN 978-5-8459-1679-2 **в продаже**

MICROSOFT ASP.NET 4 С ПРИМЕРАМИ НА C# 2010 ДЛЯ ПРОФЕССИОНАЛОВ 4-Е ИЗДАНИЕ

*Мэтью Мак-Дональд
Адам Фримен
Марио Шпушта*



www.williamspublishing.com

Книга известных специалистов в области технологий .NET представляет собой учебное и справочное пособие для разработчиков .NET-приложений, использующих новую версию ASP.NET 4. Настоящее издание было полностью обновлено и дополнено с учетом последней версии ASP.NET, и теперь включает описание ASP.NET MVC, ASP.NET AJAX 4, ASP.NET Dynamic Data и Silverlight 3. Предложенный авторами практический подход к изложению материала не является простым повторением документации, а позволяет сконцентрироваться на решении конкретных задач, связанных с разработкой веб-приложений разного уровня сложности. Глубина изложения материала превращает эту книгу в незаменимый источник информации для разработчиков приложений ASP.NET 4.

ISBN 978-5-8459-1702-7 **в продаже**

С# 4 И ПЛАТФОРМА .NET 4 ДЛЯ ПРОФЕССИОНАЛОВ

Кристиан Нейгел
Билл Ивьен
Джей Глинн
Карли Уотсон
Морган Скиннер



www.dialektika.com

ISBN 978-5-8459-1656-3

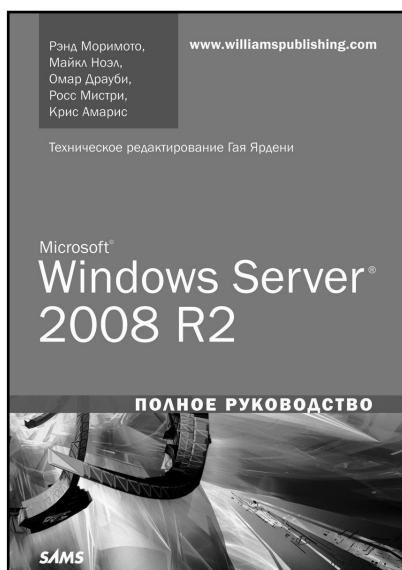
Книга известных специалистов в области разработки приложений с использованием .NET Framework посвящена программированию на языке С# 2010 в среде .NET Framework 4 и в предшествующих версиях. Книгу отличает простой и доступный стиль изложения, изобилие примеров и рекомендаций по написанию высококачественных программ. Подробно рассматриваются такие вопросы, как основы языка программирования С#, организация среды .NET, работа с данными, написание Windows- и веб-приложений, взаимодействие через сеть, создание веб-служб и многое другое. Немалое внимание уделено проблемам безопасности и сопровождения кода. Тщательно подобранный материал позволит без труда разобраться с тонкостями использования Windows Forms и построения веб-страниц. Читатели ознакомятся с работой в Visual Studio 2010, а также с применением различных технологий, встроенных в .NET. Книга рассчитана на программистов разной квалификации.

в продаже

MICROSOFT WINDOWS SERVER 2008 R2

Полное руководство

***Рэнд Моримото,
Майкл Ноэл,
Омар Драуби,
Росс Мистри,
Крис Амарис***



www.williamspublishing.com

В книге приводится исчерпывающее описание всех аспектов Windows Server 2008 R2, включая Active Directory, сетевые службы, безопасность, переход на Windows Server 2008 R2 из Windows Server 2003/2008, администрирование, отказоустойчивость, оптимизация, обнаружение неполадок, ключевые прикладные службы и многое другое.

Авторы очень точно отмечают основные усовершенствования Windows Server 2008 R2 и предлагают развернутое описание всех инноваций Windows Server 2008 R2, начиная с виртуализации Hyper-V и заканчивая DirectAccess и улучшениями компонента Failover Clustering. В каждой главе предлагаются многочисленные полезные советы и трюки, основанные на сотнях внедрений, которые позволяют максимально эффективно решать бизнес-задачи с помощью Windows Server 2008 R2. Книга рассчитана на пользователей и администраторов средней и высокой квалификации.

ISBN 978-5-8459-1653-2 в продаже