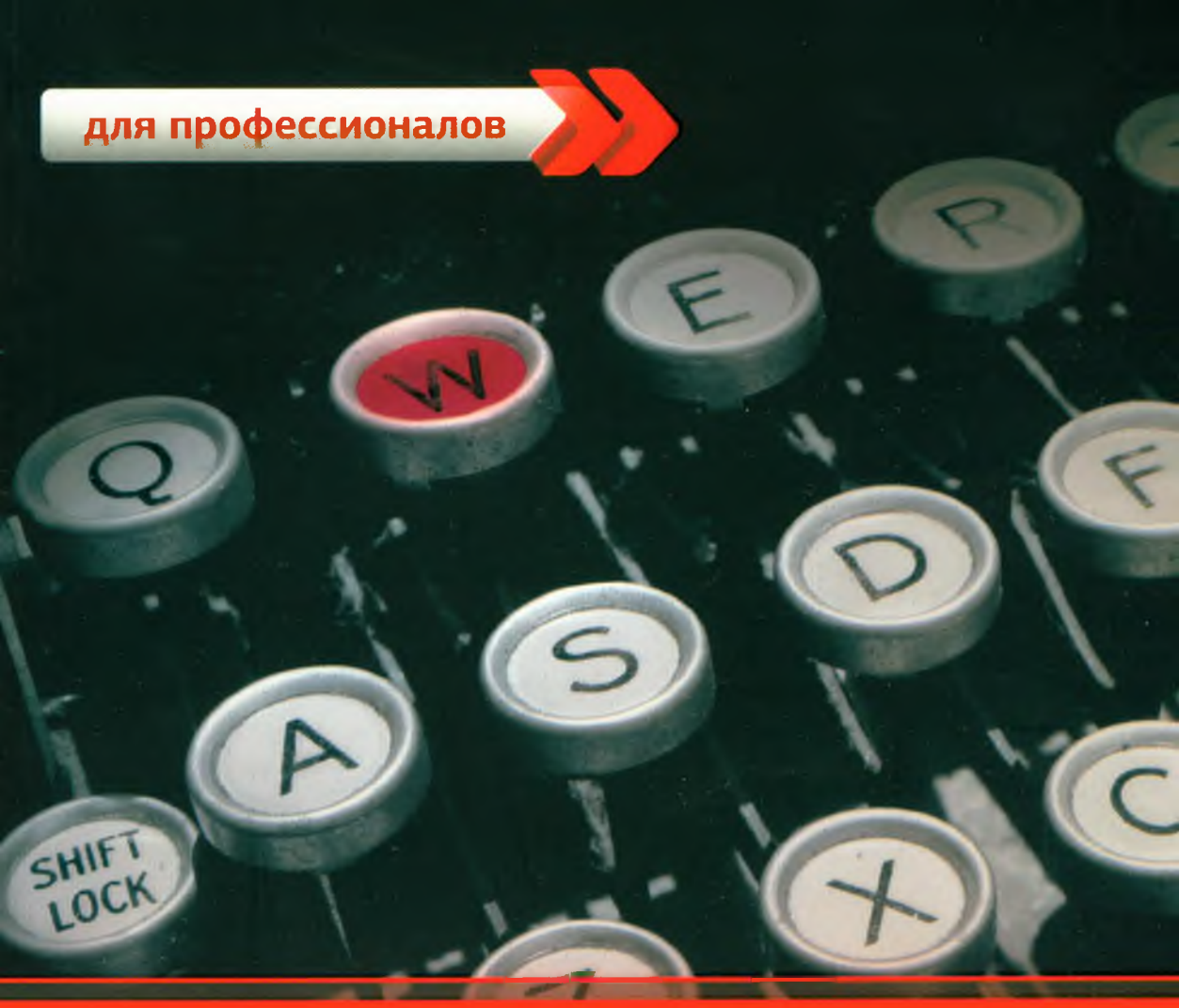
для профессионалов



WordPress

ДЛЯ ПРОФЕССИОНАЛОВ

РАЗРАБОТКА И ДИЗАЙН САЙТОВ

Б. Уильямс Д. Дэмстра Х. Стэрн

PROFESSIONAL WordPress®

DESIGN AND DEVELOPMENT

Second Edition

Brad Williams
David Damstra
Hal Stern

WILEY
John Wiley & Sons, Inc.

WordPress

ДЛЯ ПРОФЕССИОНАЛОВ

РАЗРАБОТКА И ДИЗАЙН САЙТОВ

Б. Уильямс
Д. Дэмстра
Х. Стэрн

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2014

ББК 32.988.02-018
УДК 004.738.5
У36

Уильямс Б., Дэмстра Д., Стэрн Х.

У36 WordPress для профессионалов. — СПб.: Питер, 2014. — 464 с.: ил. — (Серия «Для профессионалов»).

ISBN 978-5-496-00948-5

Эта книга, выходящая во втором издании, поможет вам стать экспертом в разработке сайтов на базе платформы WordPress. WordPress является самой популярной в мире бесплатной CMS-системой, однако большинство разработчиков используют только базовые функции WordPress, не углубляясь в профессиональную веб-разработку на ее основе. Вместе с тем если использовать WordPress по максимуму, на его базе можно создавать проекты любого уровня сложности и дизайна.

В книге подробно описана система CMS, ее основные функциональные элементы, внутренняя работа кода и структуры данных. Рассказывается о разработке собственных дизайн-тем, использовании плагинов и написании расширений, настройке и оптимизации крупных ресурсов, работающих на этой системе. Издание содержит большое количество примеров и готового кода, который можно использовать в своих проектах.

Книга адресована широкому кругу разработчиков: от тех, кто стремится выполнить тонкую настройку темы WordPress, до более опытных разработчиков, знакомых с разработкой плагинов.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988.02-018
УДК 004.738.5

ISBN 978-1118442272 англ.
ISBN 978-5-496-00948-5

Краткое содержание

Об авторах 18

Благодарности 19

Введение 20

Глава 1. Первая запись 26

Глава 2. Обзор кода 50

Глава 3. Работаем с WordPress локально 72

Глава 4. Обзор ядра 90

Глава 5. Цикл (Loop) 108

Глава 6. Управление данными 140

Глава 7. Пользовательские типы записей, пользовательские таксономии и метаданные 156

Глава 8. Разработка плагинов 182

Глава 9. Разработка тем 255

Глава 10. Multisite 311

Глава 11. Агрегация контента 342

Глава 12. Взаимодействие с пользователем 368

Глава 13. Статистика, масштабируемость, безопасность и спам 404

Глава 14. WordPress как система управления контентом 440

Оглавление

Об авторах **18**

Благодарности **19**

Введение **20**

Для кого эта книга 20
Что включает в себя эта книга 21
О структуре этой книги 21
Что необходимо для этой книги 24
Обозначения 24
Исходный код 25
Опечатки 25

Глава 1. Первая запись **26**

Что такое WordPress? 26
Популярность WordPress 28
Текущее состояние 28
О сообществе 30
WordPress и GPL 30
Контент и обсуждение 32
WordPress как система управления контентом 32
Создание обсуждения 34
Начало работы 34
Возможности хостинга 35

Установка «Сделай сам» ... 37
Установка файлов WordPress ... 37
Конфигурирование базы данных ... 40
Завершение ... 46
Администрируем в первый раз ... 46
Первая запись ... 48
Резюме ... 49

Глава 2. Обзор кода ... 50

Загрузка ... 50
Откуда загрузить ... 50
Доступные форматы ... 51
Архив версий ... 51
Структура папок и файлов ... 52
Настройка WordPress ... 53
Файл wp-config.php ... 54
Расширенные параметры wp-config ... 56
.htaccess ... 62
Файл .maintenance ... 67
Пользовательская площадка wp-content ... 68
Плагины ... 68
Темы ... 69
Загрузки и директория медиафайлов ... 69
Директория Upgrade ... 70
Персональные директории ... 70
Резюме ... 71

Глава 3. Работаем с WordPress локально ... 72

Преимущества локальной работы ... 72
Типичный цикл внедрения ... 73
Почему так много этапов? ... 74
Инструменты для администрирования компонентов ... 75
Установка инструментов для разработки ... 75
Добавление WordPress в локальную установку ... 77
Детали настройки ... 78
Управление древом документов веб-сервера ... 78
Информации для отладки ... 81
Работа с локальной и рабочей базой данных ... 83
Создание имен виртуальных локальных серверов ... 83
Разработка тем и плагинов локально ... 86

Внедрение локальных изменений ... 87
Резюме ... 89

Глава 4. Обзор ядра ... 90

Что есть в ядре? ... 90
Использование ядра как справочника ... 92
Встроенная документация ... 92
Поиск функции ... 93
Исследуем ядро ... 96
Устаревшие функции ... 99
Кодекс WordPress и ресурсы ... 100
Что такое Кодекс? ... 100
Использование Кодекса ... 100
Справочник по функциям ... 102
API WordPress ... 103
Битва за Кодекс ... 105
Не взламывайте ядро! ... 105
Почему нет? ... 106
Альтернативы взламыванию ядра ... 106
Резюме ... 107

Глава 5. Цикл (Loop) ... 108

Понимание цикла ... 109
От параметров запроса к SQL ... 110
Понимание контента в WordPress ... 112
Помещение цикла в контекст ... 112
Процесс цикла ... 114
Теги шаблона ... 116
Часто используемые теги шаблона ... 117
Параметры тегов ... 118
Индивидуальная настройка цикла ... 118
Использование объекта WP_Query ... 119
Построение произвольного запроса ... 120
Разбиение на страницы в цикле ... 123
Использование query_posts( ) ... 124
Использование get_posts( ) ... 126
Сброс запроса ... 127
Больше чем один цикл ... 129
Сложные запросы ... 130
Глобальные переменные ... 131
Данные записи ... 132

Данные автора ... 133
Данные пользователя ... 134
Данные среды ... 134
Глобальные переменные или теги шаблона? ... 136
Работа вне цикла ... 136
Резюме ... 139

Глава 6. Управление данными ... 140

Схема базы данных ... 140
Детали таблицы ... 142
Таблицы контента WordPress ... 143
Таблицы таксономии WordPress ... 144
Класс базы данных WordPress ... 146
Простые запросы базы данных ... 146
Сложные операции с базой данных ... 147
Работа с ошибками ... 149
Прямое управление базой данных ... 151
Резюме ... 155

Глава 7. Пользовательские типы записей, пользовательские таксономии и метаданные ... 156

Понимание данных в WordPress ... 156
Что такое пользовательский тип записи? ... 157
Регистрация пользовательского типа записей ... 157
Определение ярлыков типа записи ... 163
Работа с пользовательскими типами записи ... 164
Файлы шаблона записи пользовательского типа ... 165
Особые функции типа записи ... 166
Таксономия WordPress ... 168
Предустановленные таксономии ... 168
Структура таблиц таксономии ... 169
Понимание соотношений в таксономии ... 169
Построение собственных таксономий ... 170
Обзор пользовательских таксономий ... 170
Создание индивидуальных таксономий ... 170
Определение ярлыков пользовательской таксономии ... 174
Использование пользовательской таксономии ... 175
Метаданные ... 177
Что такое метаданные? ... 177
Добавление метаданных ... 178
Обновление метаданных ... 179

Удаление метаданных ... 179
Возвращение метаданных ... 180
Резюме ... 181

Глава 8. Разработка плагинов ... 182

Компоновка плагина ... 183
Создание файла плагина ... 183
Создание заголовка плагина ... 184
Лицензия плагина ... 184
Функции активации и деактивации ... 185
Интернационализация ... 186
Определение путей ... 189
Безопасность плагина ... 190
Временные значения (Nonces) ... 191
Валидация и очистка данных ... 192
Знай свои зацепки: действия и фильтры ... 195
Действия и фильтры ... 196
Популярные зацепки-фильтры ... 197
Популярные зацепки-действия ... 199
Настройки плагина ... 201
Сохранение параметров плагина ... 201
Массив параметров ... 202
Создание меню и подпунктов меню ... 203
Создание страницы параметров ... 206
Интеграция с WordPress ... 214
Создание метаполя ... 214
Сокращенные коды ... 218
Создание виджета ... 219
Создание консольного виджета ... 224
Создание произвольных таблиц ... 225
Деинсталляция плагина ... 227
Создание плагина для примера ... 228
Публикация в директории плагинов ... 247
Ограничения ... 247
Загрузка плагина ... 247
Создание файла readme.txt ... 248
Установка SVN ... 251
Публикация в директории плагинов ... 253
Выпуск новой версии ... 253
Резюме ... 254

Глава 9. Разработка тем 255

Зачем использовать тему? 255
Установка темы 256
Установка по FTP 257
Установщик темы 257
Что такое тема? 258
Файлы шаблона 258
CSS 258
Изображения и ресурсы 259
Плагины 259
Создание собственной темы 259
Темы проектов или дочерние темы 260
Что искать в стартовой теме 261
Создание своей темы. Начало 262
Основной файл: Style.css 262
Показываем контент: Index.php 264
Отображение контента различными способами: index.php 265
Создание своей темы: DRY 266
header.php 266
footer.php 268
sidebar.php 268
Отклонения от нормы: условные теги 269
Создание своей темы: отображение контента 270
Индивидуализация домашней страницы: front-page.php 271
Отображение старых записей: archive.php 273
Отображение одной рубрики: category.php 275
Отображение записей по метке: tag.php 277
Другие архивные шаблоны 278
Как показать отдельную запись: single.php 278
Отображение страницы: page.php 280
Отображение приложений к записи: attachment.php 280
Иерархия шаблонов 281
Создание собственной темы: дополнительные файлы 282
Разберемся с ошибками 404: 404.php 282
author.php 284
comments.php 285
Добавление функциональности в шаблоны: functions.php 286
search.php 289
searchform.php 290
Другие файлы 291

Шаблоны произвольных страниц ... 292
Когда использовать шаблоны произвольных страниц ... 292
Как использовать шаблоны произвольных страниц ... 293
Шаблоны страниц в Twenty Eleven ... 294
Другие расширения темы ... 295
Управление меню ... 295
Области виджетов ... 298
Форматы записей ... 299
Настройки темы ... 300
Тонкий настройщик темы ... 301
Иерархия тем и дочерние темы ... 301
Темы класса «премиум» и другие каркасы тем ... 306
Тема Bones ... 307
Тема Carrington ... 308
Тема Genesis ... 308
Тема Hybrid Core ... 308
Roots ... 309
Тема StartBox ... 309
Тема Thematic ... 309
Резюме ... 310

Глава 10. Multisite ... 311

Что такое Multisite? ... 311
Терминология Multisite ... 312
Отличия ... 312
Преимущества Multisite ... 313
Активация Multisite ... 313
Работа в сети ... 315
Консоль администратора сети ... 315
Создание сайтов и управление ими ... 315
Работа с пользователями и ролями ... 317
Темы и плагины ... 317
Настройки ... 318
Привязка домена ... 318
Кодирование для Multisite ... 319
Идентификатор блога ... 319
Общие функции ... 319
Создание нового сайта ... 323
Меню администратора сети ... 327
Параметры Multisite ... 329
Пользователи в сети ... 335

Суперадминистраторы ... 338
Сетевой статус ... 339
Схема базы данных Multisite ... 339
Специфические таблицы Multisite ... 340
Специфические таблицы сайтов ... 340
Резюме ... 341

Глава 11. Агрегация контента ... 342

Привлечение внимания ... 343
Кнопки социальных сетей ... 345
Делимся контентом ... 346
Кнопки, значки или и то и другое? ... 347
Простые значки социальных сетей ... 348
Сбор внешнего контента ... 349
Интеграция видео с YouTube ... 350
Интеграция Twitter ... 351
Google Maps ... 354
Интеграция Facebook ... 355
Универсальные данные XML ... 355
Временные объекты ... 358
Реклама ... 360
Монетизация вашего сайта ... 361
Размещение рекламы ... 362
Личная жизнь и история ... 365
Резюме ... 367

Глава 12. Взаимодействие с пользователем ... 368

Принципы взимодействия с пользователем ... 369
Единая навигация ... 369
Элементы графического дизайна ... 372
Упрощение поиска контента ... 373
Время загрузки сайта ... 374
Использование JavaScript ... 376
Простота использования и ее проверка ... 377
Структурирование информации ... 379
Как сделать ваш сайт легко обнаруживаемым ... 381
Дублирование контента ... 383
Обратные ссылки и отклики ... 385
Метки и сайты с общим контентом ... 386
Как веб-стандарты помогают обнаружить ваши данные ... 387
Семантический HTML ... 387

Валидный HTML ... 389
Микроформат ... 390
HTML5 ... 393
CSS3 ... 395
Поиск по вашему сайту ... 396
Слабые стороны поиска по умолчанию ... 396
Альтернативные и полезные плагины ... 398
Доступ с мобильных устройств и адаптивный веб-дизайн ... 399
Оставьте их в покое ... 400
Легкие версии для мобильных устройств ... 400
Адаптивный дизайн ... 401
Резюме ... 403

Глава 13. Статистика, масштабируемость, безопасность и спам ... 404

Счетчики статистики ... 404
AWStats ... 405
Google Analytics ... 408
Плагин WordPress JetPack ... 410
Управление кэшем ... 412
Сложность системы WordPress ... 413
Кэширование и оптимизация работы веб-сервера ... 414
Кэширование объектов WordPress ... 417
Временный кэш ... 418
Кэш запросов MySQL ... 419
Выравнивание нагрузки на ваш сайт WordPress ... 420
Работа со спамом ... 422
Модерация комментариев и CAPTCHA ... 423
Автоматизация обнаружения спама ... 424
Обеспечение безопасности сайта WordPress ... 425
Обновления ... 425
Сокрытие информации о версии WordPress ... 426
Ограничение количества попыток входа в систему ... 427
Использование надежных паролей ... 427
Изменение префикса таблицы ... 428
Перемещение файла конфигурации ... 428
Перемещение директории с контентом ... 428
Использование функции «Секретный ключ» ... 429
Принудительное использование SSL при входе в систему и администрировании . 430
Разрешения Apache ... 430
Имя пользователя и пароль MySQL ... 431
Рекомендованные плагины для обеспечения безопасности ... 431

Использование ролей в WordPress ... 435
Роль: Подписчик ... 436
Роль: Участник ... 436
Роль: Автор ... 436
Роль: Редактор ... 437
Роль: Администратор ... 437
Роль: Суперадминистратор ... 437
Обзор ролей ... 437
Дополнительные роли ... 439
Резюме ... 439

Глава 14. WordPress как система управления контентом ... 440

Управление контентом ... 440
Рабочие процессы и делегирование ... 442
Пользовательские роли и делегирование ... 443
Рабочий процесс ... 444
Организация контента ... 446
Поддержка тем и виджетов ... 447
Домашние страницы ... 449
Страницы избранных объектов ... 450
Иерархия контента ... 453
Интерактивные свойства ... 457
Форумы ... 457
Формы ... 457
Электронная коммерция ... 458
Другие системы управления контентом ... 459
Интеграция WordPress ... 459
Где не стоит использовать WordPress ... 460
Резюме ... 461

Моей жене, партнеру, лучшему другу — Эйприл Уильямс. Ты никогда не узнаешь, насколько много ты для меня значишь. Спасибо, что терпишь мое занудство и всегда поддерживаешь меня.

— Брэд Уильямс

Любящей меня жене Холли и моим детям — Джеку, Джастину и Иону. Спасибо за вашу любовь и поддержу.

— Дэвид Дэмстра

Тоби, чья терпимость от проекта к проекту только возрастает.

— Хэл Стерн

Об авторах

Брэд Уильямс — один из основателей WebDevStudios.com, один из организаторов подкаста WP Late Night, а также соавтор книг «Professional WordPress» (WordPress для профессионалов) и «Professional WordPress Plugin Development» (Разработка профессиональных плагинов WordPress). Брэд занимается разработкой веб-сайтов более 15 лет, а в течение последних 5 лет его работа сосредоточена на технологиях с открытым программным кодом, таких как WordPress. Брэд проводил презентации на различных конференциях WordCamp в разных частях страны и участвовал в подготовке встречи WordPress в Филадельфии и конференции WordCamp там же. Вы можете следить за деятельностью Брэда онлайн — он ведет личный блог по адресу http://strangework.com и пишет в Твиттере (@williamsba).

Дэвид Дэмстра — вице-президент отдела профессиональных услуг в CU*Answers, организации, обслуживающей кредитный союз. Дэвид руководит группой разработчиков, занимающихся созданием веб-сайтов и веб-приложений для финансового сектора. Группа, которой руководит Дэвид, использует WordPress в качестве основы для многих веб-проектов. Также у Дэвида есть сертификат компании Zend по PHP5. Посетите сайт http://ws.cuanswers.com, где Дэвид рассматривает аспекты, связанные с веб-технологиями, а также лучшие образцы мировой практики в области веб-разработок, уделяя особое внимание сфере деятельности кредитного союза. Личный блог Дэвида — http://mirmillo.com, в нем он рассказывает о своей семье и пивоварении в домашних условиях.

Хэл Стерн — вице-президент крупной технологической компании, которая занимается архитектурой программного обеспечения для программируемых сетей, а также архитектурой приложений, работающих с большими объемами данных. Хэл начал вести свой блог, пытаясь наладить корпоративную коммуникацию в Sun Microsystems, и использовал платформу WordPress, чтобы делиться своими мыслями о музыке, спорте, еде и жизни в Нью-Джерси — в течение последних 5 лет. Хэл увлекся платформой WordPress, когда пытался понять, каким образом поврежденный URL почти корректно возвращает данные. В результате Хэл принял участие в написании этой книги, а также в конференции WordCamp. Хэл ведет блог на http://snowmanonfire.com и пишет в Твиттере (@freeholdhal).

Благодарности

Спасибо тебе, Эйприл, любовь всей моей жизни, за бесконечную поддержку, дружбу и терпение к моему занудству. Спасибо моим замечательным племянницам, Индиане Брук и Остин Маргарет. Благодарю все сообщество WordPress за поддержку, дружелюбие, мотивацию и руководство.

Спасибо вам, Майкл, Джейсон, Фредди и Ганнибал, за то, что вы всегда готовы оказать поддержку, оставаясь при этом в тени. И наконец, спасибо моим смешным зверушкам — Лектеру, Клариссе и коту Сквики (также известному как Китти Галор). При взгляде на ваши улыбчивые мордочки и вертлявые хвосты я и сам начинаю улыбаться.

— Брэд Уильямс

Введение

Дорогой читатель! Спасибо, что ты взял в руки эту книгу. WordPress — самое популярное на данный момент программное обеспечение для веб-сайтов, размещенных на собственном хостинге. Оно доступно как проект с открытым кодом под лицензией GPL и построено преимущественно на основе языка программирования PHP и базы данных MySQL. Любая серверная среда, поддерживающая эту простую комбинацию, позволяет запустить WordPress, что делает его перенос исключительно легким, а саму систему — простой в установке и работе. Чтобы использовать WordPress, не нужно быть системным администратором, разработчиком, экспертом в HTML или эстетствующим дизайнером.

С другой стороны, по причине того, что WordPress разрабатывался с использованием большого набора интернет-стандартов, он может быть расширен и приспособлен для огромного множества приложений. WordPress одновременно и механизм для публикации, лежащий в основе тысяч индивидуальных блогов, и движок, на котором работают широко известные веб-сайты и блоги с большими объемами информации, такие как CNN. Он был спроектирован таким образом, чтобы им мог легко пользоваться любой, но в то же время чтобы он давал широкие возможности для веб-дизайнеров и разработчиков. Имея в своем распоряжении такое многообразие приложений и возможностей, сложно понять, с чего начать, чтобы использовать всю мощь WordPress в своих целях: следует сначала изучить модели баз данных и взаимоотношения контента с метаданными или же начать с механизмов представления, которые генерируют HTML?

Эта книга создана для того, чтобы читатели смогли досконально узнать WordPress. Главное внимание в ней уделяется внутренней структуре и основному коду, а также модели данных, на основе которой этот код работает. Знание того, как что-то функционирует, нередко помогает в работе с этим объектом, его расширении или ремонте в случае поломки. Точно так же, как гонщик получает преимущество, имея базовые представления о двигателе внутреннего сгорания, аэродинамике и механике подвески автомобиля, тот, кто использует WordPress во всем его динамическом многообразии, будет гораздо более искушен в процессе, ознакомившись с физикой программного обеспечения.

Для кого эта книга

На написание этой книги нас сподвиг колоссальный разрыв между почти пустяковым усилием, необходимым для создания сайта на основе WordPress и первой записи,

адресованной миру, и гораздо более глубоким пониманием, необходимым для доведения сайта до рабочего состояния. Многие из книг, представленных на рынке, дают инструкции начинающим блогерам, объясняя читателю функции создания, конфигурирования и обслуживания сайта на основе WordPress. Нашей же целью было перекинуть мостик от специалиста, занимающегося разработками на PHP, который без усилий читает Кодекс WordPress вместо руководства, к обычному пользователю WordPress, создающему персональный сайт, интегрированный с социальными сетями и рекламными сервисами, с подогнанным под личные требования оформлением.

Проще говоря, мы надеемся обратиться к широкому кругу разработчиков: от тех, кто стремится выполнить тонкую настройку темы WordPress, до более опытных, знакомых с разработкой плагинов. Мы будем исследовать WordPress всесторонне. В этой книге мы поставили себе целью описать базовую работу функции, а затем предложить инструкции и примеры, которые демонстрируют, как «разобрать» и повторно «собрать» эту функцию, чтобы она соответствовала целому ряду задач. Те пользователи WordPress, которые не являются уверенными разработчиками на PHP, могут предпочесть пропустить раздел, предназначенный для разработчиков, тогда как программисты, которые ищут те или иные шаблоны для реализации новой функциональности WordPress, могут начать с середины и читать до конца.

Что включает в себя эта книга

Эта книга состоит из трех основных разделов. Главы 1–4 представляют собой обзор системы WordPress, ее основных функциональных элементов и содержат поверхностное описание того, что происходит, когда отображается сгенерированная WordPress страница. Главы 5–9 основываются на этом фундаменте и погружают читателя в ядро WordPress, описывая внутреннюю работу кода и структуры данных. Этот средний раздел ориентирован преимущественно на разработчиков и описывает, как раздвинуть возможности WordPress с помощью плагинов и настроить его посредством тем. Последний раздел, главы 10–14, совмещает видение разработчиком пользовательского опыта и оптимизации с требованиями по производительности и безопасности.

О структуре этой книги

Ниже приведен детальный обзор того, что можно найти в этой книге.

Глава 1 «Первая запись» содержит краткий обзор истории программного ядра WordPress; исследует некоторые популярные возможности хостинга; рассматривает вопрос, почему сообщество важно в мире, где контент имеет первостепенное значение; излагает основы самостоятельной установки и отладки WordPress.

Глава 2 «Обзор кода» начинается с изложения механизма загрузки дистрибутива WordPress и описывает его основное содержимое и раскладку файловой системы. Обзор кода сверху донизу проводит читателя от домашней страницы или URL

отдельной записи через процесс выбора записей, сбора контента и генерирования отображаемого HTML. Эта глава представляет собой карту более детального путешествия по коду в разделе, предназначенном для разработчиков.

Глава 3 «Работаем с WordPress локально» описывает множество преимуществ работы с WordPress на локальном компьютере. В этой главе также дается обзор разнообразных вариантов установки для локальной разработки на компьютерах с операционными системами Microsoft Windows или Apple. В конце рассматривается, как перенести изменения, внесенные локально, на удаленный сервер, используя различные методы внедрения.

Глава 4 «Обзор ядра» анализирует основные функции PHP, которые включает в себя движок WordPress. Она служит введением в ориентированный на разработчиков средний раздел книги, а также является основой для посвященных внедрению, интеграции и опыту частей в последнем разделе. Эта глава также описывает использования ядра как справочного руководства и объясняет, почему лучше не взламывать код ядра для получения желаемых изменений.

Глава 5 «Цикл (Loop)» является наиболее важной для читателей, ориентированных на разработку. Основной цикл WordPress приводит в действие функции создания и хранения контента в базе данных MySQL, а также извлекает соответствующие части данных для сортировки, оформления и размещения на странице, генерируя страницу, отображаемую браузером. Эта глава разбирает процессы создания, сохранения и публикации новой записи, а также отображения контента, который был сохранен в базах данных MySQL. Функции, лежащие в основе базы данных, и управление содержимым метаданных описаны более детально, чтобы дать подробный обзор внутренней работы WordPress.

Глава 6 «Управление данными» является аналогом главы 5 для MySQL. Основные функции создания, обновления и манипулирования записями в многочисленных таблицах базы данных MySQL — эта часть охватывает схему базы данных, используемую таксономию данных и метаданных и базовые взаимоотношения между элементами WordPress. Она также включает в себя обзор базовых функций запросов, используемых для выбора и извлечения контента из MySQL, формируя основу для расширений и пользовательского кода, который должен уметь анализировать индивидуальные данные, лежащие в основе сайта на WordPress.

Глава 7 «Пользовательские типы записей, пользовательские таксономии и метаданные» исследует различные типы контента и ассоциированных данных в WordPress. Охватываются вопросы регистрации и работы с пользовательскими типами записей для создания индивидуального контента в WordPress. Также разбираются пользовательские варианты таксономий с разделением на различные типы установки с примерами. Наконец, рассматриваются метаданные записи и надлежащие пути хранения произвольных данных в записях WordPress.

Глава 8 «Разработка плагинов» начинается с рассмотрения базовой архитектуры плагинов, а затем исследует интерфейсы зацепок (hooks), действий (actions) и фильтров (filters), которые интегрируют новую функциональность вокруг ядра

WordPress. Эта глава показывает вставку функций на страницы или в потоки управления контентом, а также то, как сохранять данные плагина. Примеры построения плагина с использованием простой структуры выявляют необходимую функциональность любого плагина. Эта часть также охватывает создание виджетов — простых в использовании плагинов, которые обычно размещают дополнительные изображения или контент на боковую панель; у многих плагинов также есть виджеты для более удобного управления. Публикация плагина в репозитории WordPress и возможные проблемы с конфликтом плагинов завершают дискуссию о функциональных расширениях WordPress.

Глава 9 «Разработка тем» является аналогом главы 8, посвященным отображению и рендерингу. Плагины добавляют в ядро новые возможности и функции, тогда как темы, CSS и шаблоны страниц изменяют способ подачи контента читателям. Начиная с базовой темы, эта часть охватывает написание темы, построение индивидуальных шаблонов страницы, управление меню, область виджетов, форматы записей, установку темы и использование элементов темы функциями, описанными в предыдущих частях. Эта часть завершает предназначенный для разработчиков раздел книги.

Глава 10 «Multisite» изучает популярную возможность WordPress — Multisite. Здесь рассказывается о преимуществах собственной сети Multisite, ее правильной установке, работе в Сети, создании сайтов и пользователей, управлении темами и плагинами и даже о присвоении домена. Последний фрагмент этой части исследует кодирование для Multisite и различные функции и методы, которые можно использовать.

Глава 11 «Агрегация контента» рассматривает WordPress с позиции сервисов. Если веб-сайт представляет ваш публичный образ или интернет-ресурс, он должен получать контент от различных сервисов и источников контента. Эта часть изучает интерфейсы веб-сервисов, WordPress API, ввод и вывод из WordPress и то, как заставить записи WordPress появляться на страницах Facebook.

Глава 12 «Взаимодействие с пользователем» рассматривает установку WordPress, с точки зрения обычного или потенциального читателя. Удобство использования, тестируемость и легкость нахождения информации на сайте WordPress формируют основы, делается упор на веб-стандартах для метаданных и оптимизацию поискового движка, чтобы сайт мог быть найден при надлежащем поиске Google. Если глава 11 охватывает внедрение внешнего контента на сайт WordPress, то эта глава показывает, как заставить ваш контент появляться в других местах Интернета. Наряду с доступностью контента и его передачей на мобильные устройства обсуждаются альтернативы добавления поисковой функциональности, что является одним из недостатков WordPress.

Глава 13 «Статистика, масштабируемость, безопасность и спам» касается хорошей и плохой популярности. Защита WordPress от неизбежных комментариев спамеров, а также от атак злоумышленников является ключевой составляющей конфигурирования и управления. Эта часть охватывает наиболее популярные плагины, выполняющие защитные функции и обеспечивающие безопасность. Инструменты анализа трафика указывают, насколько хорошо определенные типы контента,

рекламные кампании, варианты продвижения или ссылки привлекают читателей и как это связано с управлением трафиком.

Глава 14 «WordPress как система управления контентом» идет дальше блогов, чтобы показать, как можно использовать WordPress для управления жизненным циклом, интеграцией и распространением сетевого контента.

Что необходимо для этой книги

Необходимо хотя бы элементарное понимание HTML и некоторое знание каскадных таблиц стилей (CSS), чтобы извлечь пользу из разделов о темах и пользовательском опыте. Опыт в написании и отладке PHP кода является необходимым условием для понимания более специализированных разделов для разработчиков, хотя если вы собираетесь вносить изменения на основе примеров из этой книги, то можете использовать код как шаблон и учиться по ходу дела. Знание основ баз данных, в особенности синтаксиса и семантики MySQL, пригодится, чтобы взять максимум из главы об управлении данными, а также о разработке плагинов, которые требуют сохранения данных.

Полезно иметь интерактивную среду разработки для просмотра кода PHP или кода PHP на страницах HTML. Выбор инструментария разработчика зачастую ограничен «религиозными» и глубоко личными предпочтениями (и мы знаем множество кодеров, которые считают, что *vi* составляет среду разработки). Некоторые из инструментов, более ориентированные на пользователя, делают просмотр кода WordPress проще, если вы хотите увидеть, как функции, использованные в примерах, выглядят в ядре.

Самое важное: если вы хотите использовать образцы кода и примеры из этой книги, вам понадобится веб-сайт на WordPress. Глава 1 описывает некоторые базовые возможности хостинга WordPress, некоторые механизмы загрузки компонентов и процесс установки WordPress на домашнем компьютере или на тестовой машине.

Наконец, некоторые могут возразить, что в полной мере использовать преимущества WordPress можно только умея писать, но это заявление упускает из виду основу красоты платформы WordPress: она использует мощь печатного пресса на индивидуальном уровне. Эта книга не о том, что вы говорите (или можете сказать); она о том, как вы собираетесь донести свои идеи до Сети, как мир увидит их и будет взаимодействовать с вашим логом.

Исходный код образцов доступен для загрузки на веб-сайте Wrox:

www.wrox.com/remtitle.cgi?isbn=9781118442272

Обозначения

Чтобы помочь читателю получить как можно больше от этого текста и не терять общую линию, мы используем в книге ряд условных обозначений.

ВНИМАНИЕ

Поля, помеченные таким образом, содержат важную информацию, относящуюся к данному тексту, которую следует помнить.

ЗАМЕЧАНИЕ

Рубрика содержит замечания, советы, указания и хитрости, имеющие отношениек обсуждаемой теме.

Что касается форматирования текста:

- *Курсивом* выделяются новые термины и важные слова при первом употреблении.
- Имена файлов и фрагменты кода в тексте выглядят следующим образом: `persistence.properties`.
- Код может быть представлен двумя способами:

```
Для большинства примеров кода используется моноширинный шрифт.
Полужирное начертание применяется для выделения наиболее важных фрагментов кода
в данном контексте.
```

Исходный код

Изучая примеры из этой книги, вы можете либо перепечатывать код вручную, либо использовать файлы исходного кода, прилагаемые к этой книге. Исходный код, использованный в данной книге, можно скачать с сайта www.wrox.com. Тексты листингов для этой книги находятся на следующей странице:

www.wrox.com/remtitle.cgi?isbn=9781118442272

Большинство кодов на www.wrox.com хранятся в виде архивов .ZIP, .RAR или других аналогичных форматах, соответствующих конкретным платформам. Скачав код, просто разархивируйте его программой, которую вы предпочитаете.

Опечатки

Мы постарались приложить все усилия, что устранить ошибки из текста и кода. Однако никто не совершенен, и ошибки случаются. Если вы найдете ошибку в одной из наших книг, например опечатку или неработающий фрагмент кода, мы будем благодарны за информацию об этом. Послав нам сведения об опечатке, вы можете уберечь другого читателя от многочасового разочарования и в то же время помочь нам повысить качество представляемой информации.

Список найденных в этой книге опечаток можно найти, пройдя по ссылке:

www.wrox.com/remtitle.cgi?isbn=9781118442272

Затем следует кликнуть на ссылке Errata. На открывшейся странице вы увидите все известные опечатки.

1 Первая запись

В этой главе:

- Благодарность предшественникам платформы WordPress
- Выбор подходящей платформы для установки WordPress
- Скачивание, установка и выполнение базовой конфигурации WordPress
- Диагностика и решение общих установочных проблем

Если отображение сообщения «Привет, мир!» на соответствующем устройстве определяет наличие минимальных знаний в языках программирования, то создание первой записи является эквивалентом этого действия в мире онлайн-публикаций. В этой главе излагается краткая история WordPress, а затем изучаются некоторые возможности установки и размещения WordPress. Типичные промахи и моменты недопонимания, а также способы их устранения тоже содержатся в этой главе, что поможет вам выйти на передовую в вопросах публикации своих острот и мудрых умозаключений.

После того как вы закончили установку, конфигурацию и администрирование основных компонентов, можно приступать к изучению кода и детального описания компонентов в последующих главах. Конечно, если у вас уже есть сайт, работающий на WordPress, вы можете пропустить эту главу и сразу погрузиться в изучение кода ядра в главе 2 «Обзор кода».

Что такое WordPress?

WordPress — одна из самых популярных систем управления контентом с открытым кодом, окруженная глобальным сообществом заинтересованных пользователей, разработчиков и служб поддержки. Несмотря на возможность сравнения с TypePad, Moveable Type, Google's Blogger и проектом Apache Roller, WordPress отличается

от них наличием широкого спектра возможностей, функциональных расширений (плагинов) и эстетических элементов дизайна (тем).

С ростом возможностей самостоятельной публикации, вариантов недорогого хостинга и свободно доступных компонентов ядра, таких как базы данных MySQL, программное обеспечение для блогов следует той же тенденции, что и большинство других цифровых технологий, двигаясь от передовой дорогостоящей продукции к широко доступным бюджетным пользовательским, или «любительским», системам. WordPress — это не просто создание блога с целью иметь цифровой дневник, прикрепленный к приятно ласкающему ваше самолюбие URL. Он развился в полноценную систему управления контентом, используемую в равной мере и отдельными пользователями, и предприятиями. Этот раздел дает краткий экскурс в раннюю историю WordPress, завершающийся описанием нынешней версии и актуального пользовательского сообщества.

WordPress возник аналогично многим другим популярным пакетам программного обеспечения с открытым кодом: несколько талантливых разработчиков увидели необходимость создать мощный, но простой инструментарий на основе существующего проекта с лицензией GPL. В данном случае отправной точкой стала система b2/cafelog Мишеля Вальдриги. WordPress был выстроен как ответвление этой кодовой базы разработчиками, которых звали Мэтт Мюлленвег и Майк Литтл. Впервые WordPress появился в 2003 году на основе базы данных с открытым кодом MySQL (для сохранения данных) с PHP в качестве базового языка разработки. Вальдриги продолжает вносить свой вклад в проект, который процветает благодаря растущему сообществу пользователей и разработчиков.

Как и другие системы, написанные на PHP, WordPress является автономным в том смысле, что задачи, связанные с установкой, конфигурацией, управлением и администрированием содержатся в модулях PHP. Популярность WordPress частично обусловлена простотой этой системы, словосочетание «установка за пять минут» используется практически в любом ее описании или любой книге о ней. Помимо возможности опубликовать первую запись, WordPress был создан таким образом, чтобы расширяться и адаптироваться к различным потребностям различных людей.

В наши дни WordPress поддерживается несколькими разработчиками ядра и многими программистами, вносящими в дело свой вклад. Майк Литтл держит специализированный магазин WordPress zed1.com и время от времени делает «заплатки» (патчи) для кода. Компания Мэтта Мюлленвега *Automattic* продолжает управлять хостинг-сервисом wordpress.com, а также поддерживает разработку соответствующего инструментария для управления контентом и сайтами, включая Akismet, WordPress Multisite и Gravatar. Akismet — это надежный сервис защиты от спама, располагающийся на Automattic, со статистически невероятно низким уровнем ошибок при его определении. Ранее известные как WordPress MU, функции WordPress Multisite — в сердце системы хостинга wordpress.com и на данный момент включены в основное древо исходников WordPress. Gravatar динамически поставляет изображения, привязанные к электронным адресам, снабжая иконки многочисленными вариантами отображения. Подумайте о нем как о сервисе, позволяющем сделать

оперативную привязку изображения в вашем профиле технически и социально приемлемым.

Определение WordPress как системы управления контентом не ограничивается расположением по хронологии записей с комментариями. BuddyPress представляет собой набор тем и плагинов, которые расширяют WordPress до функциональной платформы социальных сетей, позволяя зарегистрированным пользователям посылать сообщения, общаться друг с другом, а также взаимодействовать со всем содержанием, управляемым в рамках WordPress. Таким же образом bbPress представляет собой систему на основе PHP и MySQL, спроектированную для форумов (досок объявлений), которая отличается от WordPress, но обычно интегрируется с ним.

Популярность WordPress

Эта книга основывается на WordPress 3.5, основная версия. Каждая следующая версия WordPress включает в себя улучшения, внесенные в функции администрирования и управления (Консоль (Dashboard)); в функции резервного сохранения, экспорта и импорта, а также в элементы установки и обновления. Даже если вы начинаете с несколько более ранней версии WordPress, вы сможете обновить ее до текущей версии и поддерживать актуальность. Установка и обновление рассматриваются в этой главе далее. Но насколько же популярен WordPress?

Текущее состояние

Интерес к WordPress и его использованию растет с огромной скоростью. Вы держите в руках наглядное тому подтверждение. Всего 3 года назад книг о WordPress было буквально несколько штук. Сейчас мы выпустили второе издание этой. «Популярный» — всегда понятие субъективное, но статистика делает личное восприятие более весомым. В соответствии с данными Automattic на 2011 год ежедневно создается более 100 000 сайтов на WordPress (http://en.wordpress.com/stats/). Сюда входят сайты, использующие WordPress для управления контентом, и блоги, число которых, конечно, следует уменьшить на количество пользователей, имеющих несколько копий WordPress. Но даже при такой оценке WordPress остается чрезвычайно популярным. Automattic насчитал порядка 74 миллионов сайтов на WordPress по всему миру, примерно половина из которых располагается на wordpress.com (http://en.wordpress.com/stats/). В предыдущем издании этой книги мы указывали число всего в 5 миллионов. В 2008 году официальный репозиторий плагинов WordPress вмещал более 6300 плагинов, в 2007 году это число удвоилось. На момент написания данного текста количество плагинов превышает 19 000 (http://wordpress.org/news/2012/05/plugins-refreshed/). В официальном репозитории тем WordPress находится более 1500 уникальных тем, и это не считая коммерческих вариантов и тем, созданных независимыми разработчиками.

Чтобы представить комбинации плагинов и тем во всей сложности и полноте, потребовались бы специальные научные исследования, но в то же время все они

просты в размещении, интегрировании и использовании. Это результат стабильной архитектуры и работы не менее стабильного сообщества, которое ее использует. Проще говоря, экосистема, окружающая WordPress, живет и процветает.

В августе 2011 года Мэтт Мюлленвег представил отчет о текущем состоянии WordPress, а также результаты самого первого тематического анкетирования на WordCamp в Сан-Франциско. Отчет о WordPress аналогичен глобальной переписи населения WordPress с включением информации о том, как эта система используется ежедневно. Анкетирование прошли независимые веб-разработчики, корпоративные пользователи и любители, то есть это внушительный срез, отображающий положение дел для большей части населения WordPress. Приведенные ниже выдержки показывают, насколько WordPress распространен в Интернете:

- Порядка 15 % и 1 миллиона самых посещаемых веб-сайтов используют WordPress.
- В среднем 22 из 100 новых сайтов запускаются на WordPress.
- Из репозитория плагинов было скачано более 200 миллионов плагинов.
- 18 000 ответивших на опрос представляют более 170 000 сайтов.

Ключевые моменты выступления Мюлленвега можно увидеть на wordpress.tv.

На сегодняшний день WordPress поддерживает сайты многих крупных медиакомпаний полностью или частично, включая блоги CNN, *Wall Street Journal's All Things D*, Reuters, Forbes и грубоватый, но построенный на игре слов www.icanhazcheeseburger.com. WordPress используется 500 компаниями, такими как GM, UPS и Sony. Это жизнеспособный выбор для целого ряда пользователей от международных корпораций до любимых публикой исполнителей и крупных медиакомпаний. Некоторым нужны веские аргументы, чтобы выбрать WordPress, им нужно понять, какие из «солидных парней» им пользуются. Список можно найти на «доске почета» выдающихся пользователей WordPress (http://en.wordpress.com/notable-users/).

Простота и легкость в использовании и исключительная мощь плагинов и тем делают WordPress подходящим и для информационного сайта вашей мамы, и для новостной рассылки учителя начальной школы, и для человека, увлеченного каким-то хобби. Это некоторые из успешных вариантов использования WordPress в наши дни, их можно легко найти. WordPress делают популярным именно популярные сайты. Система легко адаптируется и будет настолько простой или сложной, насколько вам необходимо. Воодушевление «низкотехнологичных» пользователей на публикацию информации в Сети и дальнейшие рассказы в кругу семьи и друзей о простоте использования WordPress и привели к имеющемуся взрывному росту его популярности.

С чего начать? wordpress.org — здесь расположены версии кода: актуальная и находящаяся в разработке. Посмотрите wordpress.org/extend, чтобы понять, где искать плагины, темы и списки пожеланий по внедрению идей и возможностей.

wordpress.org предлагает как бесплатные, так и платные услуги хостинга. По ссылке wordpress.org/hosting вы найдете список компаний, предоставляющих хостинг,

поддерживающий WordPress (нередко он включается в базовый пакет дополнительные услуги при первичной установке и конфигурированию).

О сообществе

WordPress преуспевает и растет в том числе и благодаря вкладу, вносимому сообществом. Этот процесс можно уподобить участию в спортивных соревнованиях в старших классах школы: эти игры позволяют объединить усилия и энергию игроков в рамках неформальных занятий.

WordCamp — событие, устраиваемое сообществом на локальном уровне. Сейчас они организуются в десятках городов по всему миру. Официальные WordCamp входят в список на wordcamp.org, но вы можете спокойной поискать WordCamp в ближайшем к вам крупном городе. WordCamp проводятся практически каждые выходные при участии блогеров, фотографов, писателей, редакторов, разработчиков и дизайнеров, имеющих разный опыт и уровень знаний. WordCamp — недорогой способ познакомиться с местным сообществом и нередко хорошая возможность встретить знаменитостей из мира WordPress. Загляните на wordcamp.org, чтобы узнать, когда будет проходить следующий WordCamp.

Менее структурированно, но более часто, чем WordCamp, проводятся WordPress Meetup (встречи WordPress), собирающие местных пользователей и разработчиков примерно в 200 (сравните с 40, названными в первом издании этой книги) городах. Вам необходима учетная запись на meetup.com, и, зарегистрировавшись там, вы сможете отслеживать место и время проведения на wordpress.meetup.com, чтобы знать, где и когда люди обсуждают вопросы управления контентом.

Обширный мультиязычный репозиторий с документацией находится по адресу codex.wordpress.org. Кодекс WordPress, именуемый так по наследству от древних манускриптов, представляет собой собираемый сообществом свод советов и хитростей, касающихся всех аспектов WordPress, от установки до отладки. Если вы ощущаете настоятельную потребность внести свою лепту в документацию WordPress, зарегистрируйтесь и выплесните все, что накопилось, в Кодекс WordPress. Есть надежда, что эта книга окажется для вас чем-то средним между компаньоном и путеводителем по Кодексу.

Наконец, для тех, кто вносит свой вклад, и для участников различных сообществ существуют рассылки (и их архивы). Актуальный перечень доступен по адресу: codex.wordpress.org/mailing_lists. Особенно интересными нам кажутся: wp-docs — список для тех, кто вносит свой вклад в Кодекс, и wp-hackers — список для тех, кто работает с ядром WordPress и определяет дальнейшее направление его развития.

WordPress и GPL

WordPress лицензирован в соответствии с Универсальной общественной лицензией GNU (GPL), версия 2, с которой можно ознакомиться в файле licence.txt в корне дистрибутива кода. Большинство людей не читает лицензию и просто считает WordPress проектом с открытым кодом. Однако юридические отделы корпораций

все еще боятся за свои доходы из-за «вирусной» составляющей лицензии GPL и ее применения для дополнительных кодов или контента, который добавляется к оригинальному дистрибутиву или используется вместе с ним. Большинство случаев недопонимания происходит вследствие вольного использования слов «свободный» и «авторское право» в том или ином контексте.

Авторы этой книги не являются юристами и не исполняют роли таковых в Интернете или на телевидении, так что если вы действительно хотите понять все тонкости закона об авторском праве и выяснить, что именно относится к «передаче права» на код, обратитесь к работам Лоуренса Лессига или Коури Доктороу в этих областях. Этот раздел мы включили, чтобы свести к минимуму озабоченность отделов ИТ, которым излишне усердные команды юристов могли отсоветовать использовать WordPress как систему управления контентом на предприятии. Не допускайте, чтобы это произошло и с вами. Повторяем: если WordPress приемлем для CNN и *Wall Street Journal*, двух компаний, выживающих благодаря авторским правам на содержание, то, возможно, он точно так же впишется в юридические структуры большинства корпоративных пользователей.

Основной принцип GPL гласит, что вы всегда можете получить исходный код для любого дистрибутива программного обеспечения с лицензией GPL. Если компания изменяет пакет программного обеспечения с лицензией GPL, а затем перевыпускает эту новую версию, она должна также сделать исходный код общедоступным. Вот «вирусная» природа GPL в действии. Ее цель — убедиться, что доступ к программному обеспечению и его производным никогда не ограничивается. Если вы планируете изменить ядро WordPress, а затем перевыпустить полученный код, вам необходимо убедиться, что вносимые вами изменения покрываются GPL и что код доступен в виде исходника. С учетом того, что WordPress написан на PHP, интерпретируемом языке, распространение программного обеспечения и распространение кода являют собой по сути одно и то же действие.

Ниже приведены наиболее распространенные примеры недопонимания и разъяснения, касающиеся использования WordPress в коммерческих ситуациях.

- **«Свободное программное обеспечение» означает невозможность коммерциализации его использования.** Вы можете нанять людей для работы с вашей копией WordPress, или получать деньги от рекламы на вашем сайте, или же использовать WordPress как платформу для управления контентом онлайн-магазина. Именно так работает wordpress.com, он также позволяет Google взимать плату с рекламодателей за использование его Linux-серверов. Вы можете найти темы WordPress профессионального качества с нехилым ценником или платить хостинг-провайдеру сотни и даже тысячи долларов в год за запуск MySQL, PHP, Apache и хранение программного обеспечния WordPress; и то и другое включает в себя коммерциализацию WordPress.

- **Если вы подогнали код для использования в личных целях (типы контента, политика безопасности, какие-то навигационные требования), вы должны опубликовать эти изменения.** От вас требуется сделать исходный код общедоступным только в случае перевыпуска программного обеспечения. Если вы решили

оставить эти изменения внутри компании, необходимости в перевыпуске нет. С другой стороны, если вы внесли какие-то изменения в ядро WordPress, от них выгадает все сообщество. Найти разумных работодателей, способных понять важность вклада в сообщество и ради этого ослабить авторские права и правила для сотрудников, — задача непростая. Но наличие фундамента доказывает, что другие работодатели сделали такой выбор в пользу замечательного сообщества WordPress.

- **GPL «инфицирует» контент, закладываемый в WordPress.** Контент — в том числе графические элементы тем, записи и страницы, управляемые WordPress, — отделен от ядра WordPress. Он управляется программным обеспечением, но не является его производным или частью. Однако темы являются производными кода WordPress и потому также подпадают под действие GPL, что требует от вас сделать исходный код темы общедоступным. Заметьте, что вы, тем не менее, можете взимать плату за тему, если хотите сделать ее доступной на коммерческих условиях. Повторяем: ключевым моментом является тот факт, что вы делаете исходный код доступным для любого, кто использует программное обеспечение. Если вы хотите взимать плату за использование темы, вам необходимо сделать исходный код доступным по лицензии GPL, но, как было указано ранее, устанавливая тему, пользователь, по сути, получает исходный код.

Значительно важнее, чем изучить историю WordPress и ознакомиться с его лицензией, понять, какие проблемы можно решать с его помощью и почему вам стоит решиться на это и начать получать удовольствие от его надежности и простоты. В следующем разделе WordPress рассматривается как развитая система управления контентом, а не как простой инструмент редактирования блога.

Контент и обсуждение

Километры полок в книжных магазинах заполнены томами, которые должны помочь вам в улучшении писательского слога, литературного стиля, техники ведения блога и других аспектов способности генерировать контент. Одна из целей этой книги — определить визуальные, стилистические механизмы, а также механизмы, связанные с управлением контекстом, которые вы можете использовать в WordPress для построения интересных пользовательских сообществ вокруг предлагаемой вами информации. Контекст стимулирует общение с читателями. Даже если вы интересный писатель, речь идет не только о том, что вы пишете. Есть и другие вопросы. Как люди найдут вас? Как вы можете выделиться в толпе? Как вы сделаете свой сайт индивидуальным и настроите его для своих целей: персонального или корпоративного использования, построения сообщества или достижения коммерческих целей?

WordPress как система управления контентом

Системы блогов основаны на простых операциях по управлению контентом: создать запись, поместить ее в надежное хранилище (такое, как файловая система или база

данных), отобразить отформатированный результат на основе набора каких-либо критериев типа времени или ключевых слов. С расширением богатства и многообразия контента, представленного на страницах блогов, и увеличением требований к сортировке, поиску, отбору и представлению содержания вплоть до включения метаданных и таксономии контента, граница между простым программным обеспечением, предназначенным для ведения блога одиночным пользователем, и системами управления контентом промышленного уровня стерлась.

Системы управления контентом (CMS) отвечают за создание, хранение, поиск, описание (аннотирование) и публикацию (отображение) различных типов содержимого. CMS обеспечивают выполнение стандартных задач, обычно касающихся редактирования и публикации, а также ряд действий типа одобрения и пометки контента для отправки на дополнительное редактирование или пересмотр. Консоль WordPress предоставляет такие элементы управления текущими задачами и редакторского контроля.

WordPress — не единственная широко распространенная система управления контентом.

Drupal и Joomla не менее популярны. Drupal и Joomla начинаются с репозиториев управляемого контента, они имеют дело с разнообразными типами содержимого, многочисленными авторами с разными ролями и доставкой контента клиенту, который его запросил. WordPress же, по сути, — система ведения блогов. Его конечной целью является отображение контента для читателя. Несмотря на наличие перекрывающихся функциональных областей, его можно интегрировать с другими системами управления контентом. Этот процесс подробно описан в главе 14.

WordPress заработал репутацию *добросовестной* системы управления контентом благодаря возможности расширения и строгому отделению содержимого от его отображения. Позволяя себе некоторые вольности со схемой «модель-представление-поведение» (Model-View-Controller), можно провести следующее сравнение. WordPress разделяет: 1) персистентный слой MySQL как модель данных; 2) управляемый темой пользовательский интерфейс и функции отображения; 3) архитектуру плагинов, которая добавляет функциональности при отображении данных. Чрезвычайно важно, что WordPress хранит контент в необработанном виде как данные, введенные пользователем или приложением через WordPress API. Контент не форматируется, не прогоняется через шаблоны и не становится частью эскизов до момента формирования страницы, что делает функции, генерирующие финальный HTML-код, особо мощными. В то же время модель данных, используемая WordPress, имеет обширный набор таблиц для управления категориями (таксономиями), метками контента (фолксономиями), информацией об авторах, комментариями и другими перекрестными значениями. Схема базы данных WordPress, делающая все это возможным, изучается в главе 6.

Хотя такое устройство наделяет WordPress невероятной мощью и гибкостью в качестве системы управления контентом, оно также требует знания того, как соотносятся персистентность данных и управление выводом данных (изучение WordPress с точки зрения этих функциональных терминов и сподвигло нас на написание этой книги).

Создание обсуждения

> Обсуждение — король, контент — просто тема для разговора.
>
> — *Коури Доктороу*

Жизнеспособность CMS определяется полезностью контента. Даже богатейшее содержимое и прекрасно управляемые процессы имеют низкую отдачу, если никто не использует результат этой работы. Недостаточно установить программное обеспечение, сделать несколько записей и ждать, что мир покажется на вашем виртуальном пороге. Вам нужно создать то, что Тим О'Рейлли называет «архитектурой участия». На ваш сайт читателей приведут социальные сети, отзывы и приемы, гарантирующие его появление в результатах, отображаемых поисковыми системами. Дизайн, продвижение и графические элементы, помноженные на качество контента, воодушевят читателей на активное участие.

Посмотрим на проблему с точки зрения читателя. Как вас найти и услышать в мире, где существуют миллионы сайтов (на многих из которых нет ничего, кроме «первой записи»)? Наверняка читатели вашего Twitter захотят заглянуть и на ваш сайт, а сайт на WordPress может обновлять ленту в Twitter. И наоборот: обновления в Twitter могут появляться на боковой панели WordPress, соединяя ультракороткую информацию с более вдумчивой. Если вы активны на Facebook, вы можете импортировать записи на страницы этой соцсети, а читатели Facebook будут приносить трафик на ваш сайт. Если в своих записях вы затрагиваете специфические или таинственные области или пишете о чем-то подробно, поиск в Google по соответствующим словам должен выводить читателей прямо на вас, чтобы они присоединились к обсуждению. Глава 11 «Агрегация контента» затрагивает вопрос помещения в WordPress контента из других социальных медиасистем и систем управления контентом, а в главе 12 «Взаимодействие с пользователем» рассматривается, как наиболее широко распространить содержимое из WordPress.

Начало работы

Перед тем как вы серьезно займетесь внешним видом, стилем или контентом сайта, сначала ваш сайт должен обрести свой дом (несмотря на предыдущую дискуссию о WordPress и системах управления контентом, мы будем часто использовать термины «сайт» и «WordPress» в одном значении, что приведет к взаимозаменяемости этих понятий, преимущественно для удобства и краткости). Среди факторов, влияющих на выбор:

- **Стоимость.** Услуги бесплатного хостинга ограничивают вас как разработчика и частенько не позволяют вам получать деньги от рекламных сервисов. Более дорогие предложения могут включать в себя лучшую техническую поддержку, больший объем для хранения данных и пропускную способность или же обеспечивать несколько баз данных для дополнительных приложений.

- **Контроль.** Какой инструментарий предоставляется вам для управления базами данных MySQL, файлами, входящими в копию WordPress, и другими типами контента? Если вы хотите иметь возможность работать на уровне SQL или управлять MySQL через интерфейс командной строки, вам необходимо убедиться, что хостинг-провайдер поддерживает эти возможности.
- **Сложность.** Вы можете установить веб-сервер Apache с интерпретатором PHP, MySQL и дистрибутив WordPress самостоятельно.С другой стороны, большинство хостинг-провайдеров упростили процесс установки настолько, что у вас вряд ли возникнут какие-то сложности с этим. Если вы ожидаете, что вам понадобится техническая поддержка на уровне операционной системы и выше, найдите провайдера (возможно, это ваш собственный ИТ-отдел), который предоставляет такую поддержку в разумные сроки.

Этот раздел дает краткий обзор некоторых возможностей хостинга, объясняет основы самостоятельной установки WordPress и завершается описанием проблемных случаев, когда WordPress и MySQL игнорируют друг друга или что-то идет не так.

Возможности хостинга

Существуют три основных варианта хостинга для WordPress, каждый из которых представляет собой компромиссный вариант между сложностью администрирования и степенью контроля. Самый простой и наиболее популярный способ — использовать wordpress.com, бесплатный хостинг, запущенный компанией Automattic с использованием multi-site-версии WordPress (изначально WordPress MU). Здесь вы можете устанавливать тему и плагины через Консоль, но только в варианте включения или выключения тех возможностей, которые предустановлены. Кроме того, у вас нет доступа к базам MySQL и основному коду и вы не можете интегрировать WordPress с другими системами. Вы можете перенаправить один из ваших URL на wordpress.com, но если вы хотите контролировать все — от кода до используемого URL, вам, возможно, следует поискать платную альтернативу. Бесплатный вариант может быть разумным первым шагом, но здесь подразумевается, что вы собираетесь препарировать установленную вами копию.

Базовый список платных хостинг-провайдеров вы можете найти на www.wordpress.org, включая платные возможности на wordpress.com. У большинства из них есть последняя или близкая к последней версия основного кода WordPress в виде пакета для установки вместе с MySQL и веб-сервером. Третий вариант — установить все на ваших собственных серверах. Если ваши серверы обладают возможностями для хостинга и у вас есть доступ администратора, это эквивалентно установке по типу «Сделай сам».

Для WordPress необходим веб-сервер с поддержкой PHP, возможность переписывать URL (ссылки) и копия MySQL. Apache — наиболее популярный вариант для внешнего интерфейса WordPress, поскольку он обеспечивает интерпретацию PHP через `mod_php` и переписывание URL в `mod_rewrite`. Растет интерес к lighttpd (Lighty) в качестве альтернативы Apache, хотя здесь переписывание URL требует больших усилий. Наконец, вы можете использовать IIS 7.0 от Microsoft в качестве

веб-сервера с модулем `URL_rewrite`. Акцент на переписывании URL имеет место из-за поддержки WordPress «красивых» перманентных ссылок на записи в блоге, позволяющей создавать древо URL, организованное по дате, категориям, меткам или метаданным. Эти мнемонические или удобные для чтения пользователем URL конвертируются в запросы для базы данных MySQL, что позволяет извлечь нужный контент WordPress исходя из названий или ключевых слов как составляющей основного цикла WordPress, подробно описанного в главе 5. Ваш веб-сервер решает, должна ссылка быть проанализирована WordPress или же ее следует отнести к специфическому HTML файлу исходя из того, что содержится в файле `.htaccess`, а правила переписывания URL гарантируют, что содержимое будет интерпретировано корректно. Технически переписывание URL не требуется для установки WordPress, но неплохо иметь эту возможность, поскольку она дает невероятную гибкость в вопросах представления и наименования URL для контента. Проектирование и использование перманентных ссылок более подробно рассматривается в главе 2, но держите это требование в уме, выбирая платформу для WordPress.

До сего момента MySQL упоминался только походя, однако краткий обзор требований MySQL завершает список условий, выдвигаемых к хостингу. Имеет смысл оговорить некоторую терминологию и разницу между программным обеспечением MySQL, экземплярами баз данных и копиями WordPress, использующими MySQL. Установив и сконфигурировав MySQL, вы получаете работающую полноценную реляционную систему баз данных. Ее не обязательно конфигурировать на той же машине, на которой расположен веб-сервер, некоторые хостинг-провайдеры создают горизонтально расширяемые фермы MySQL параллельно интерфейсам веб-сервера. *Экземпляр* MySQL, запущенный на сервере, может поддерживать множество *баз данных*, каждая из которых будет иметь уникальное имя. При установке WordPress вам необходимо знать *имя* базы данных MySQL, зарезервированной под ваш контент, хотя эта информация может быть сгенерирована автоматически и сконфигурирована для вас, если ваш провайдер поддерживает WordPress и MySQL как интегрированный пакет. WordPress создает некоторое число *таблиц* реляционных данных в названной базе данных для каждого создаваемого вами сайта.

Неразбериха может возникнуть в результате номенклатуры и сложности. Вы (или ваш хостинг-провайдер) можете запустить несколько экземпляров MySQL на нескольких серверах, и вам необходимо будет знать, где располагается ваша база данных. Поскольку каждый экземпляр MySQL может поддерживать несколько баз данных, а каждая база данных содержит группу таблиц, можно запускать приложения на основе разных MySQL на одной и той же хостинговой платформе, используя один экземпляр MySQL или даже одну базу данных MySQL.

Если вы хотите держать несколько сайтов на WordPress на одном сервере, вы можете использовать один экземпляр базы данных MySQL для всех сайтов, если вы сконфигурируете WordPress так, чтобы он различал *имена таблицы базы данных MySQL* в рамках одной базы данных MySQL. Это простая возможность конфигурации, которая рассматривается в следующем разделе, она подчеркивает разницу между множественными наборами таблиц в базе данных и множественными базами данных для определенных приложений.

После обеспечения необходимого фундамента самое время установить и запустить код. Даже если вы используете хостинг-провайдер, который сам устанавливает WordPress и MySQL, полезно знать, как взаимодействуют компоненты на сервере: на тот случай, если вам потребуется отследить проблему при разработке плагина.

Установка «Сделай сам»

Знаменитая легендарная «установка WordPress за пять минут» — реальность, если все сконфигурировано и выполнено без ошибок. Этот раздел описывает шаги, которые обычно скрыты, если вы используете услуги провайдера с пакетной установкой, и акцентирует внимание на некоторых нестыковках между копиями WordPress и MySQL.

Процесс установки достаточно простой (при условии, что веб-сервер и сервер MySQL уже работают): скачать пакет WordPress и установить в древо директорий на веб-сервере, после чего перейти к URL верхнего уровня и завершить конфигурацию. Одно (сложное) предложение описывает процесс в полной мере.

Возможно — и даже рекомендуется — устанавливать полностью функциональную версию WordPress на ваш ноутбук или десктоп, в особенности если вы собираетесь работать с ядром, разрабатывать плагины или иным образом вносить изменения, которые могут породить неприятные ошибки при тестировании на публичном сайте. В операционную систему Mac OS X входит веб-сервер Apache (с PHP и переписыванием URL); скачайте MySQL с www.mysql.com или используйте конфигурацию в виде пакета, такую как MAMP (www.mamp.info, включающую phpMyAdmin), и вы получите самодостаточную лабораторию по разработке и тестированию вашего приложения. Что касается других платформ, то у XAMMP (www.apachefriends.org) есть аккуратно интегрированный вариант, работающий на базе Windows, MacOS и Linux. Иметь все под одной крышей — мощный способ проверки ошибочных вариантов, как вы увидите в следующих двух главах. Больше информации о работе с WordPress локально предоставляется в главе 3.

Установка файлов WordPress

Если вы скачали код WordPress с www.wordpress.org, то вы получите архив zip (или tarball), который разворачивается в директорию `wordpress`. Первая часть установки WordPress заключается в копировании кода в корневую директорию веб-сервера; убедитесь, что он находится в нужном месте, — это критически важно. Если вы ошибетесь, то весьма вероятно, что ваш сайт будет открываться по URL типа `http://example.com/wordpress` и нужно либо начинать все по новой, либо делиться этой уродливой ссылкой с друзьями. Если то, чего вы хотите, — это отделить WordPress от другого содержимого вашего сайта или изолировать несколько разделов, выбор пути вашей файловой системы чрезвычайно важен.

Выберите директорию, в которую вы хотите установить WordPress. Чаще всего это корневая директория веб-сервера, а если вы используете хостинг-провайдер, то это,

возможно, будет поддиректория `public_html` в древе файлов. Если вы используете пакетную установку, снабженную меню, которое спрашивает о конечной директории, убедитесь, что вы выбрали директорию верхнего уровня (да, убедитесь, что она уже существует, это важно!). Если вы копируете файлы с локальной машины на веб-сервер с использованием FTP-клиента, убедитесь, что выбран верный пункт назначения. По умолчанию после копирования файла zip на сервер и его распаковки все окажется в поддиректории `wordpress`, так что если вы хотите, чтобы адрес вашего сайта на WordPress был `http://example.com`, а не `http://example.com/wordpress`, переместите файлы директорией выше перед тем, как двигаться дальше. Существует возможность конфигурации, при которой копия WordPress находится в поддиректории относительно директории верхнего уровня, так что нет ничего фатального, если вы разместите WordPress в не самом удачном месте файловой системы. Этот вопрос рассматривается в конце данного раздела.

После того как WordPress установлен, проводник по файловой системе должен показать вам что-то вроде изображенного на рис. 1.1 с `index.php` и шаблоном файла `wp-config-sample.php`. Это и есть система WordPress, которая эффективно заработает в рамках интерпретатора веб-сервера PHP.

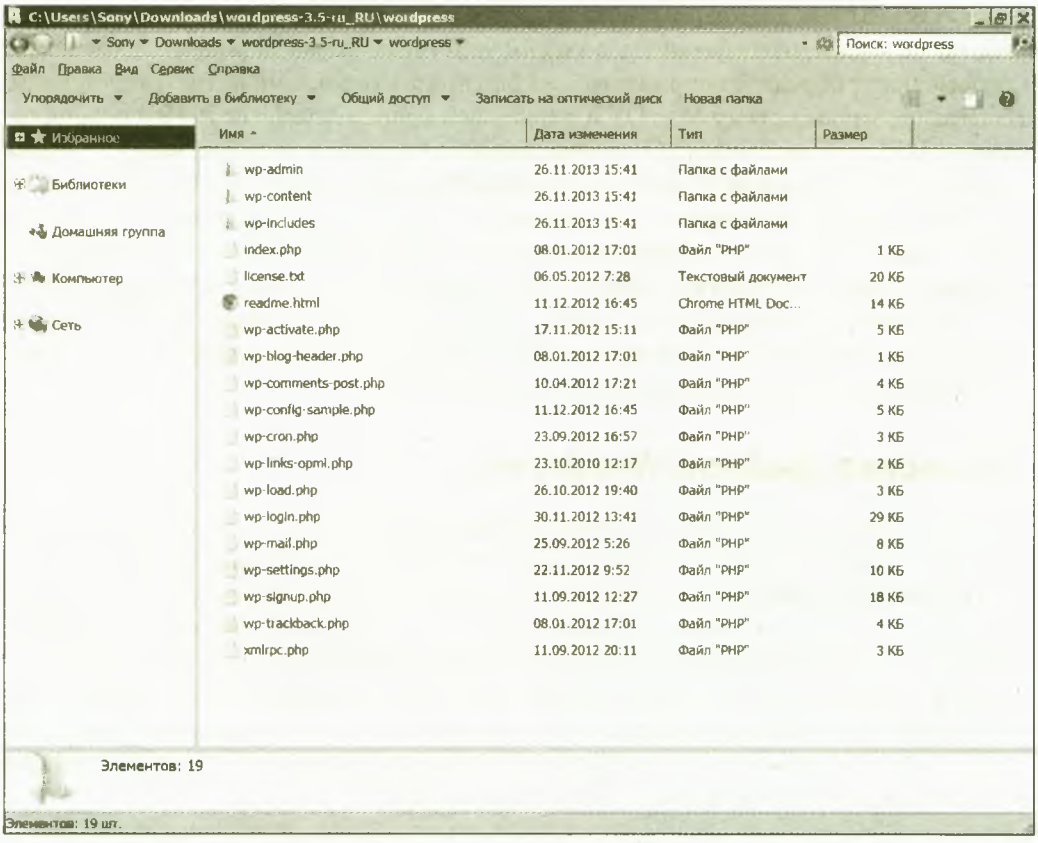


Рис. 1.1. Чистая, но несконфигурированная копия WordPress

Если вы осуществляете установку вручную, в этот момент вы захотите создать собственный файл `wp-config.php`, отредактировав предоставленный шаблон и сохранив его в директории WordPress верхнего уровня. В качестве альтернативы вы можете перейти по ссылке на сайт, после чего WordPress заметит, что конфигурационного файла не существует, и выдаст диалоговые окна, подобные представленным на рис. 1.2 и 1.3, которые вы должны будете заполнить. Вам понадобится вписать имя базы данных MySQL, имя пользователя базы данных, а также придумать префикс

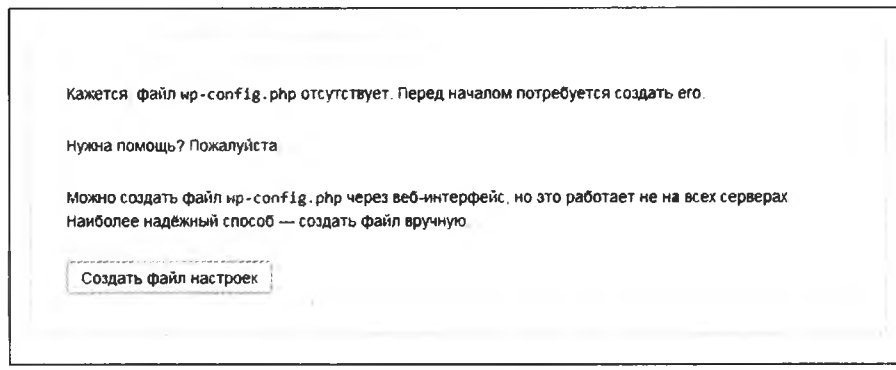


Рис. 1.2. WordPress создаст новый файл wp-config, если его не существует

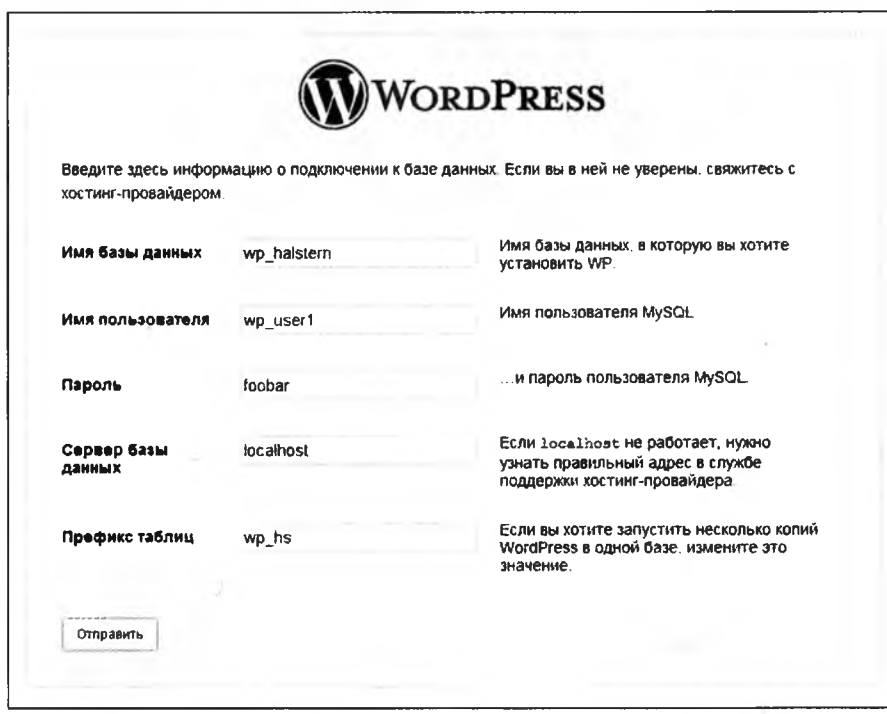


Рис. 1.3. Диалоговое окно конфигурирования базы данных

для таблиц базы данных WordPress (отличный от префикса `wp_` по умолчанию). Эти низкоуровневые детали — суть следующего раздела, посвященного конфигурированию базы данных. Если вы используете хостинг с пакетной установкой, то, возможно, вы не увидите этот шаг, поскольку файлы WordPress будут извлечены, информация по базе данных MySQL включена в конфигурационный файл автоматически, без участия конечного пользователя.

Что делать, если по ссылке уже есть HTML или другой контент и вы хотите добавить WordPress на уже существующий сайт? Куда поместить существующие файлы, зависит от того, что, по вашему мнению, должен увидеть пользователь, пройдя по ссылке. Чтобы использовать WordPress в качестве системы управления контентом, как это описано здесь, лучше всего сохранить существующее содержимое и конвертировать его в новые записи или страницы, сделав предыдущий сайт контекстом для сайта на WordPress. В качестве альтернативы вы можете установить WordPress в поддиректорию, сохранив существующий файл `index.html`, и перенаправлять читателей к новому контенту с помощью кнопки или ссылки на домашней странице. Главное — не пустить дело на самотек: если у вас есть файл `index.html` и вы устанавливаете WordPress, то получается два файла — `index.php` и `index.html`, из которых пользователи будут видеть только один в зависимости от конфигурации Directory Index на веб-сервере с вашим сайтом. Что именно делать с имеющимся содержимым, следует решать в зависимости от того, насколько много трафика оно привлекает на сайт. Если страницы отвечают за привлечение трафика с поисковых систем, возможно, не стоит менять имеющиеся ссылки, которые могут быть кэшированы, и лучше установить WordPress в поддиректорию. Если вы хотите использовать WordPress в качестве оболочки для существующего контента, переместите его и включите переписывание URL или переадресацию для страниц, ставших частью WordPress.

Если вы используете пакетную установку от хостинг-провайдера или же создали файл `wp-config.php` вручную, а затем перешли к URL верхнего уровня, WordPress должен был закончить создание таблиц баз данных, создать пользователя-администратора для вашей копии WordPress и установить пароль по умолчанию, как это показано на рис. 1.4. Убедитесь, что вы сменили имя пользователя на что-то отличное от admin.

После успешной установки вы увидите окно, как на рис. 1.5, подтверждающее, что ваши пять минут знаменитой установки истекли.

Следующий раздел рассматривает конфигурирование связки MySQL и WordPress более подробно, и ее имеет смысл прочитать, даже если от одной мысли об SQL вас начинает бить нервная дрожь. Если бы торопитесь, то можете пропустить следующий раздел и отправиться прямо к пункту **Завершение**.

Конфигурирование базы данных

Если ваш хостинг-провайдер запустил для вас базу данных MySQL и создал пользователя, загляните в конечный файл `wp-config.php`, чтобы получить эту

информацию. Она необходима для изучения MySQL, описываемого в этом разделе, и пригодится, если вы столкнетесь с проблемами с MySQL позже. В файле содержится комбинация имени пользователя и пароля, так что обращайтесь с ней бережно, как вы обращаетесь с другой информацией такого рода. С другой стороны, если вы собираетесь продвигаться дальше по маршруту «Сделай сам», сейчас вы узнаете, на что похожи недоразумения при попытке совместить компоненты и что значит оцепенеть от ужаса при возникновении ошибки.

Теоретически установка MySQL для WordPress не представляет собой ничего интересного. Убедитесь, что MySQL установлена и запущена, создайте в MySQL пользователя WordPress, а затем создайте от имени этого пользователя базу данных,

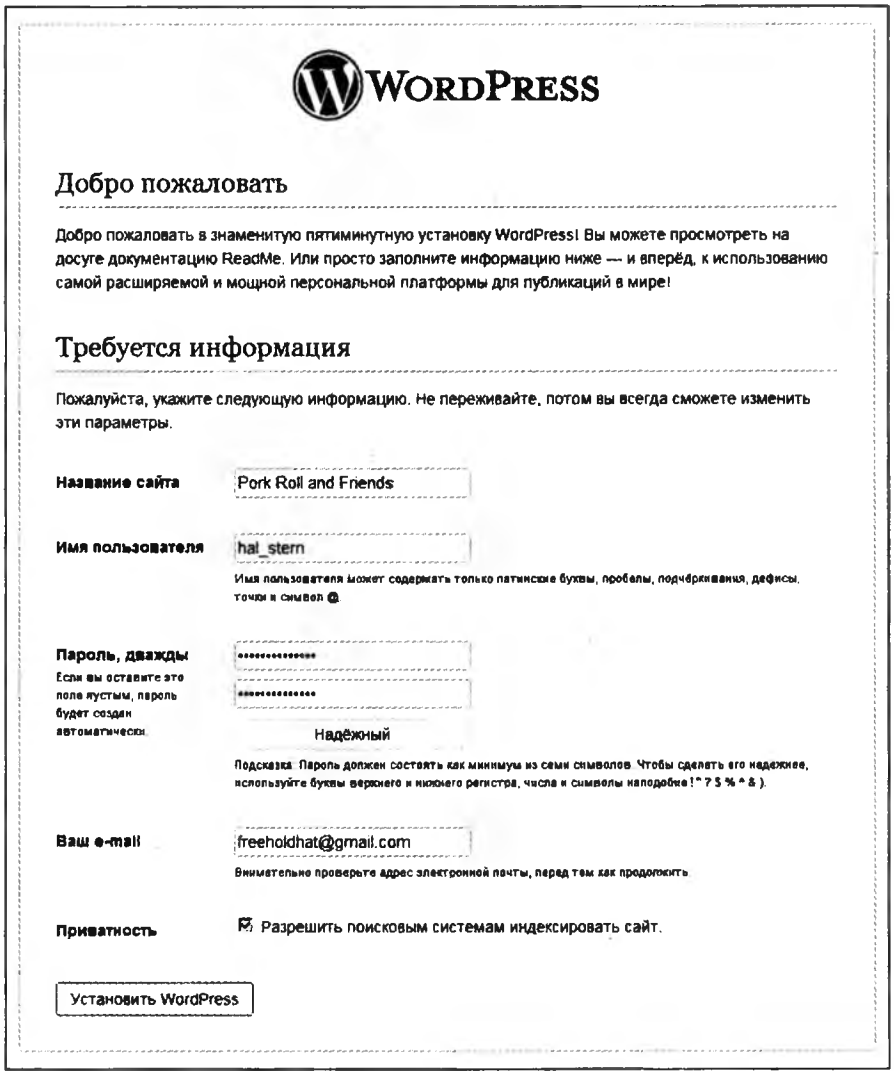


Рис. 1.4. Заполните информацию о сайте и пользователе-администраторе

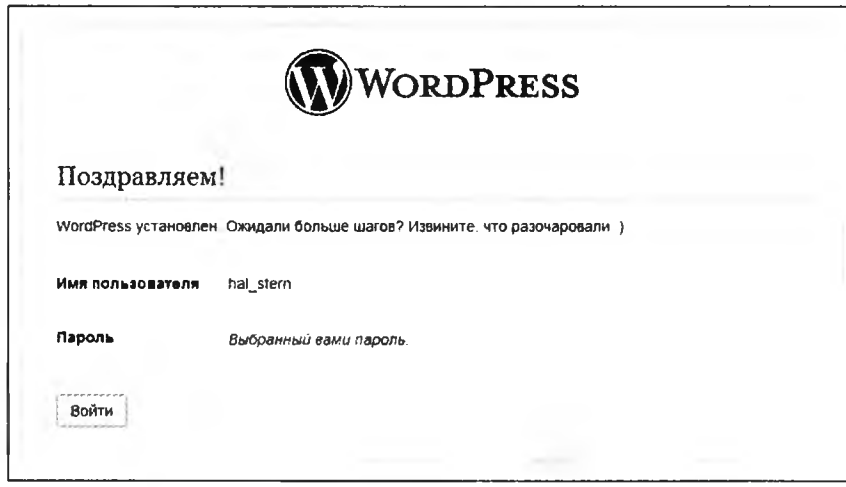


Рис. 1.5. Административная информация в конце успешной установки

которая будет содержать таблицы WordPress. Для решения этих задач вы можете использовать командную строку MySQL или инструменты типа phpMyAdmin или Chive, но держите в голове, что у MySQL есть собственный набор пользователей и данных им прав, что определяется вашей (или хостинг-провайдера) операционной системой. После установки MySQL создает таблицу пользователей и их прав по умолчанию, добавляя привилегированного пользователя `root` в систему Unix, не имеющего отношения к root-пользователю самой системы Unix. Однако если вы пытаетесь подключиться к вашей копии MySQL как привилегированный пользователь MySQL, это можно сделать только с локального хоста — той же самой машины, на которой запущен MySQL. Если вы хотите узнать больше о правах MySQL, и таблице, управляющей выдачей пользователям эти права, и о том, как управлять пользователями MySQL, обратитесь к «Руководству по пользованию MySQL» (http://dev.mysql.com/doc/) и найдите в нем разделы, касающиеся безопасности базовых учетных записей MySQL.

Для пользователей или баз данных WordPress не существует сводов правил наименования. Хостинг-провайдеры обычно добавляют имя пакета или информации из учетной записи, чтобы определять пользователей, соарендующих базу данных MySQL. Повторяем: можно иметь несколько баз данных, которыми владеет один и тот же (или разные) пользователь MySQL, на одной копии сервера базы данных MySQL. В примере, показанном на рис. 1.3, `wp_` используется как префикс и для имен пользователей, и для имен баз данных, что как минимум указывает администратору базы данных, что они относятся к WordPress.

Что может пойти не так с WordPress и MySQL? Ниже приведены три основные причины ошибок при установке. Обратите внимание, что все эти три условия должны быть соблюдены одновременно.

1. **Веб-сервер не может найти MySQL.** Либо имя хоста MySQL в файле `wp-config.php` указано некорректно, либо веб-сервер ищет локальный экземпляр

MySQL и не может открыть сокет-соединение с ней. Вот простой пример. При локальном запуске MySQL на MacOS MySQL создает сокет `/tmp/mysql.sock` для локальных соединений, но PHP-код WordPress ищет `/var/mysql/mysql.sock` через PHP-модуль движка MySQL. Создайте символьную ссылку с одного сокета на другой:

```
# ln -s /tmp/mysql.sock /var/mysql/mysql.sock
```

Актуальный путь в файловой системе к локальному сокету MySQL — функция конфигурации базы данных; при ее запуске создается локальный сокет. Где именно движок PHP и где, соответственно, любое приложение на основе PHP ищет сокет, зависит от конфигурации PHP. Если вы хотите выяснить, в чем именно заключается несоответствие, придется потратить некоторое время на непростую отладку в стиле `printf()`.

Отредактируйте файл `wp-includes/wp-db.php`, он содержит набор функций, который определяет соединение с базой данных WordPress. Если вы видите сообщение «Ошибка установки соединения с базой данных» («Error establishing a database connection») во время установки, вставьте `echo(mysql_error())` — оператор, позволяющий увидеть выявленную ошибку вместе с базовым сообщением, как это показано на рис. 1.6:

```
if (!$this->dbh) {
echo(mysql_error());
    $this->bail(sprintf(/*WP_I18N_DB_CONN_ERROR*/"
    <h1>Error establishing a database connection</h1>
```

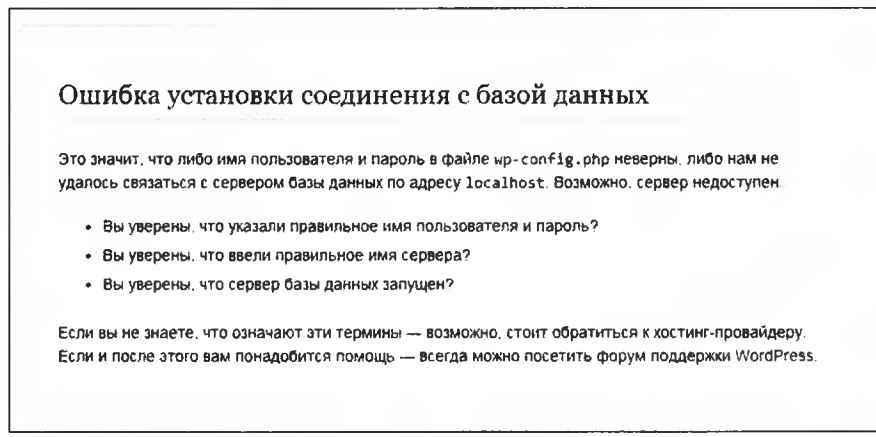


Рис. 1.6. mysql_error( ) сообщает о проблеме с сокетом

Функция `mysql_error()` является функцией библиотеки PHP, выдающей ошибку, сгенерированную последней вызванной функцией MySQL.

2. **WordPress находит MySQL, но не может войти.** Чаще всего неверно введено имя пользователя или пароль для MySQL, особенно если это произвольное имя, сгенерированное хостинг-провайдером. Дважды проверьте эти данные и убе-

дитесь, что они верно отображаются в файле `wp-config.php`. Вы также можете столкнуться с проблемой аутентификации пароля при использовании комбинации MySQL 4.1 или MySQL 5.0 с серверами PHP, имеющими определенные особенности. Они могут поддерживать только старую схему хэширования паролей MySQL 4.0. В таком случае используйте функцию `OLD_PASSWORD()` для хэширования пароля пользователя WordPress в формате, совместимом с предыдущими версиями. Вот эта магическая формула SQL (в командной строке MySQL или в окне SQL в случае MAMP):

```
SET PASSWORD FOR user@host = OLD_PASSWORD('password') ;
```

В этом варианте `user@host` — имя пользователя базы данных WordPress и имя хоста базы данных, а `password` — (буквально) пароль, который вы внесли в конфигурационный файл.

3. **WordPress подключается к MySQL, но не может выбрать базу данных.** То, что веб-сервер может подключиться к серверу базы данных с информацией о пользователе базы данных WordPress, не означает, что там обязательно есть база данных, доступная для этого пользователя. Это еще один сценарий, который лучше всего диагностируется с помощью `mysql_error()` посредством помещения этого оператора в `wp-dp.php`, где определяется ошибка выбора:

```
function select($db) {
    if (!@mysql_select_db($db, $this->dbh)) {
        $this->ready = false;
echo(mysql_error());
        $this- >bail(sprintf(/*WP_I18N_DB_SELECT_DB*/'
        ... <h1>Can't select database</h1>
        ..
```

Если после вставки оператора `mysql_error()`, как это описано выше, попытки закончить установку приводят к появлению окна с ошибкой типа показанного на рис. 1.7, то либо база данных MySQL под соответствующим пользователем создана не была, либо пользователь базы данных не имеет прав на ее использование. Дважды проверьте, что думает на эту тему MySQL, используя следующую команду:

```
% /usr/local/mysql/bin/mysql -u wp_user1 -p
Enter password:
Welcome to the MySQL monitor. Commands end with; or \g.
Your MySQL connection id is 174
Server version: 5.1.37 MySQL Community Server (GPL)
mysql> show databases;
+--------------+
| Database     |
+--------------+
| information_schema |
| test               |
+--------------+
2 rows in set (0.00 sec)
```

Если после того, как вы вошли как пользователь базы данных MySQL, вы не видите базу данных MySQL, — возможно, она была создана root-пользователем MySQL и у пользователя MySQL для копии WordPress нет соответствующих прав на доступ или внесение изменений. Если у вас есть доступ привилегиро-

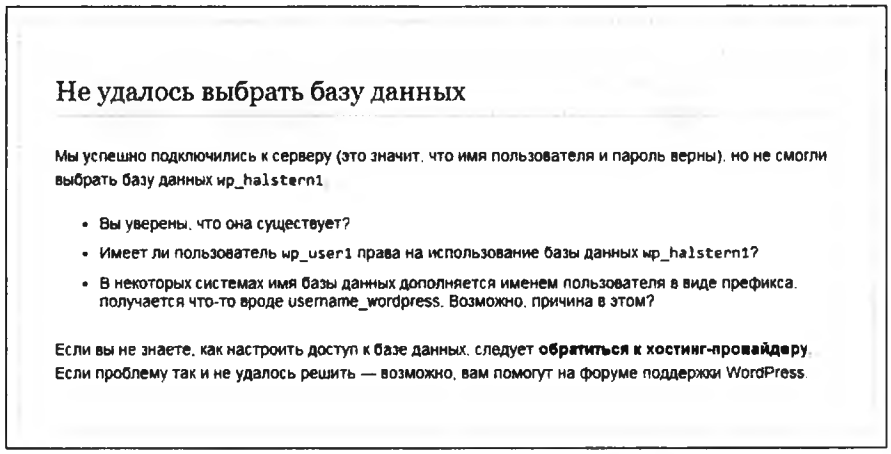


Рис. 1.7. Ошибка выбора базы данных MySQL

ванного уровня к MySQL или же достаточные пользовательские привилегии для создания новых баз данных в этой копии MySQL, несложно создать базу данных, авторизовавшись с помощью командной строки:

```
mysql> create database wp_halstern;
Query OK, 1 row affected (0.00 sec)
```

Повторяем: важно различать пользователей операционной системы, пользователей MySQL и пользователей WordPress. Пользователи MySQL определены в базе данных и наделены правами создавать базы данных, копаться в таблицах и иными способами генерировать полезные данные. Пользователи WordPress существуют в таблицах базы данных WordPress, созданных во время установки. Они имеют привилегии и значение, только если вы авторизуетесь в WordPress.

Получив чистую копию WordPress, вы увидите набор таблиц, названных в соответствии с префиксами таблиц, установленных в `wp-config.php`. Повторяем: это легко проверить, используя команду MySQL:

```
mysql> use wp_halstern; show tables;
Database changed
+-----------------+
| Tables_in_wp_halstern    |
+-----------------+
| wp_hs_comments           |
| wp_hs_links              |
| wp_hs_options            |
| wp_hs_postmeta           |
| wp_hs_posts              |
| wp_hs_term_relationships |
| wp_hs_term_taxonomy      |
| wp_hs_terms              |
| wp_hs_usermeta           |
| wp_hs_users              |
+-----------------+
10 rows in set (0.00 sec)
```

В этом примере вы установили префикс таблицы базы данных `wp_hs_`; если позже вы добавите другую копию WordPress, используя того же пользователя базы данных и тот же экземпляр, вы сможете просто установить другой префикс и получить два сайта, сосуществующих в одной таблице базы данных. Вы глубже изучите схему и погрузитесь в вопросы использования десяти основных таблиц базы данных WordPress в главе 6. Сейчас же, если вы успешно соединились с MySQL, то можно перейти к финальной чистке и начать администрировать.

Завершение

На данный момент ваша база данных MySQL установлена и запущена. Для вашего контента создан дом, и веб-сервер успешно выполняет основной код WordPress. Осталась еще всего лишь пара вещей для обсуждения.

Администрируем в первый раз

После того как вы закончили процесс установки, авторизуйтесь с логином и паролем, которые вы задали в соответствии с рис. 1.4, и вы увидите базовую Консоль WordPress, как это показано на рис. 1.8.

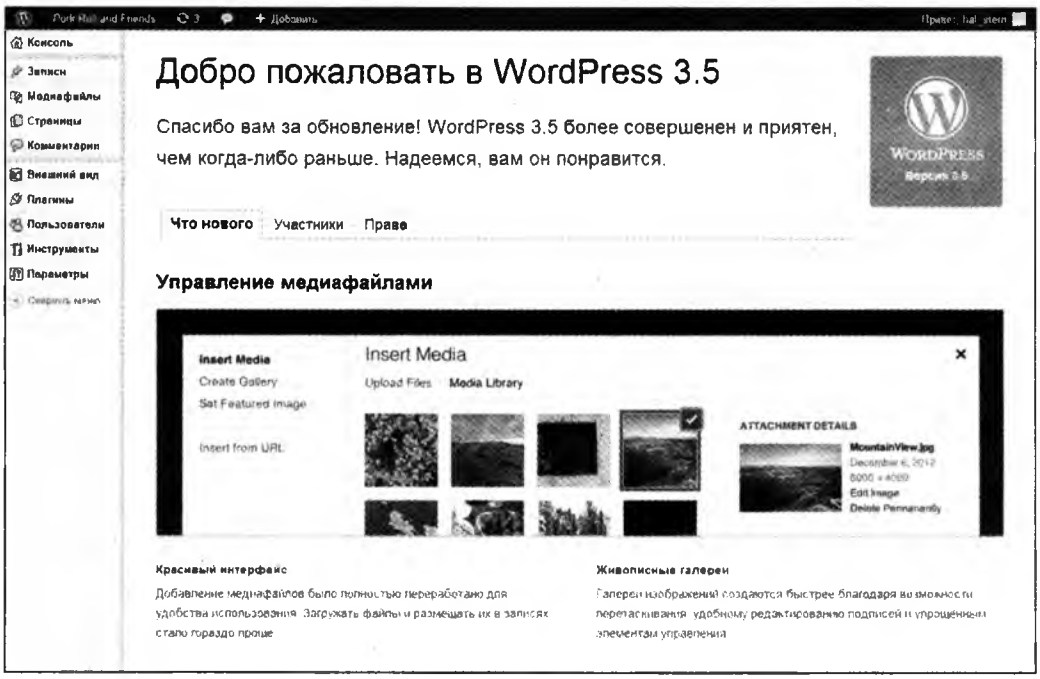


Рис. 1.8. Вид Консоли при первой авторизации

Если кнопка Войти (Log In) не перенаправляет вас в Консоль или если вы сначала попали на URL верхнего уровня на вашем сайте, либо щелкните по ссылке Войти,

либо перейдите в директорию `wp-admin` (`example.com/wp-admin`), набрав путь вручную, чтобы увидеть диалоговое окно авторизации. Авторизация на сайте приводит вас в Консоль WordPress, чрезвычайно простую в своей мощи и в то же время богатую в своей сложности и предлагаемых возможностях.

Что именно вы будете делать дальше в Консоли, зависит от того, насколько вас удовлетворяет базовая установка. Если, как в предыдущем примере, вы получили устаревшую версию WordPress, щелкните по кнопке Обновить (Update), чтобы произвести обновление до последнего дистрибутива «на месте». В дополнение к возможности самоустановки WordPress обладает функциями самообновления (в `wp-admin/includes/update.php`, если вы их ищете).

Вы можете решить изменить некоторые из базовых установок, таких как имя базы данных или имя пользователя базы данных MySQL, хотя изменить установленные по умолчанию `root@localhost` вы сможете, только имея полный контроль над веб-сервером и сервером базы данных. В конфигурационном файле также есть записи, касающиеся «ключей безопасности», предназначенных для обеспечения большей безопасности при использовании куки в браузере. Ключи безопасности более подробно обсуждаются в главе 11. Редактирование файла `wp-config.php` приведет к немедленным изменениям. Например, смена префикса таблицы базы данных заставит WordPress приписать значения новому набору таблиц и создать копию «с чистого листа». Внесите изменения и вернитесь к URL верхнего уровня: вы обнаружите новую информацию о пользователе-администраторе, не потеряв авторизации на стартовой странице Консоли, как это показано на рис. 1.8. Старые таблицы не удаляются из MySQL, поэтому вам не нужно проводить чистку вручную.

В этот момент, если вы хотите, чтобы URL отличался от места установки WordPress, вы можете выбрать в Консоли раздел Параметры (Settings) и Общие (General), после чего изменить ссылки и для адреса верхнего уровня, и для директории установки WordPress. Если вы хотите отделить URL сайта и директорию WordPress, убедитесь, что вы переместили файл `index.php` в требуемое место, а затем отредактируйте последнюю строку, чтобы добавить правильный путь к поддиректории WordPress.

Перед тем как создавать первую запись, неплохо также установить структуру Постоянных ссылок, чтобы все, что вы пишете, соответствовало выбранным вами правилам присвоения имен, благодаря чему читателям будет относительно легко находить контент и делиться им. Как и следовало ожидать, эта настройка также находится в разделе Параметры в Консоли. Варианты присвоения имен постоянным ссылкам и их воздействие на производительность и схему базы данных более подробно рассматриваются в следующей главе.

Вне зависимости от того, потратили вы действительно пять минут или же пять часов на выискивание несоответствий в именах хостов, именах пользователей и настройках базы данных, теперь вы готовы опубликовать первую запись в своем блоге.

Первая запись

В успешно установленной копии WordPress уже есть первая запись и комментарий в опубликованном виде, что подтверждает слаженное взаимодействие всех частей и снабжает ваш сайт контентом «по умолчанию». Если вы готовы добавить свои первые слова, используйте панель QuickPress (Быстрая публикация) в Консоли (возможно, сначала вам потребуется пропустить всплывающую подсказку по работе с новым сайтом) или отправляйтесь на вкладку **Записи (Posts)** и щелкайте там по **Добавить новую (Add New)**, чтобы перейти во встроенный редактор WordPress. На рис. 1.9 показана работа с записью в Быстрой публикации. После успешной публикации записи Консоль обновится.

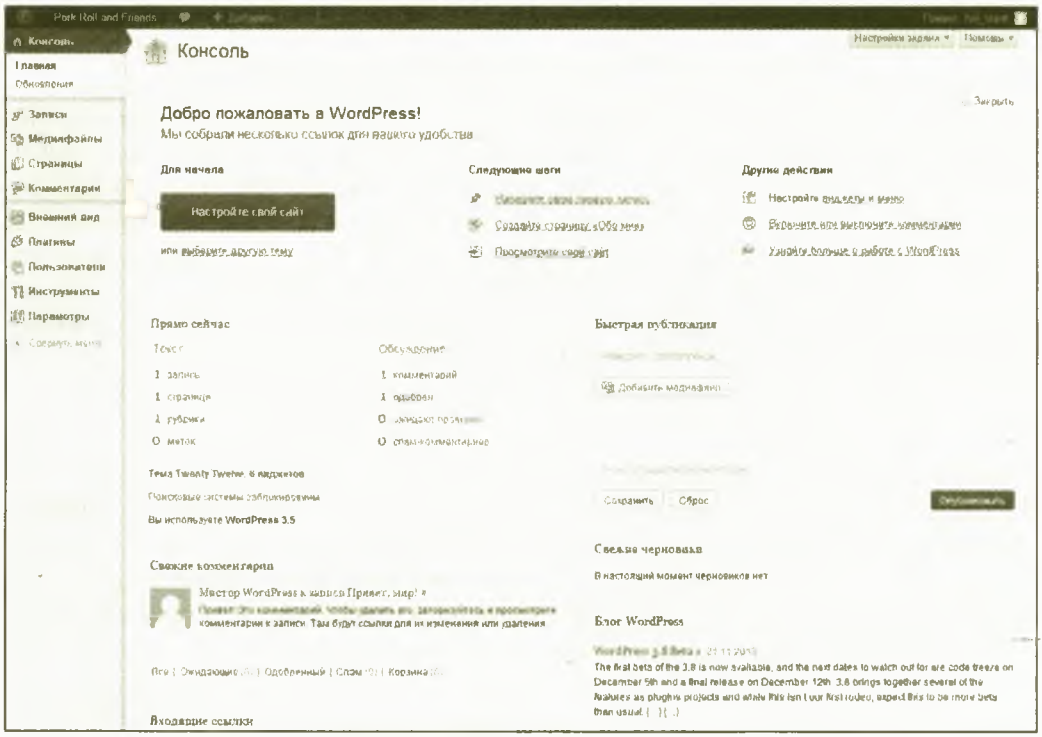


Рис. 1.9. Публикация с панели Быстрой публикации

Если ваши вкусы более старомодны, вы всегда можете создать содержимое в своем любимом редакторе, а затем вставить его в панель редактирования. Будьте осторожны с системами обработки текста типа WYSYIWIG, такими как Microsoft Word или OpenOffice, если хотите скопировать в WordPress HTML, поскольку он будет снабжен дополнительными тегами и информацией по стилям. Наконец, разнообразные автономные редакторы, такие как Illumnix от Ecto, публикуют в WordPress с использованием Atom Publishing Protocol или XML-RPC. Настройки

удаленно публикуемых записей, как вы уже догадались, можно найти в параметрах Консоли во вкладке Чтение (Writing options).

Нажмите Опубликовать (Publish), чтобы, наконец, сказать: «Привет, мир!» Многочисленные подсистемы создали эту панель редактирования, сохранили содержимое в базе данных, сгенерировали и сохранили относящиеся к ней метаданные и затем выдали симпатичный HTML. Большая часть из видимых пользователю деталей управляется через Консоль. Некоторые функции будут рассмотрены в следующих главах.

Резюме

В этой главе мы рассказали, как WordPress дожил до сегодняшнего дня, включив в нее краткий урок истории и обзор текущей популярности системы. Своим «подъемом» в виртуальной реальности WordPress отчасти обязан простоте процесса установки. В следующих главах вы погрузитесь в ядро WordPress и сможете воспользоваться преимуществами, обеспечиваемыми его расширяемостью, удобным дизайном и функциями.

2

Обзор кода

В этой главе:

- Загрузка WordPress
- Настройка `wp-config.php` и `.htaccess`
- Изучение директорий `wp-content`
- Активация профилактического режима в WordPress

WordPress представляет собой пакет программного обеспечения, который включает в себя группы файлов исходного кода, выполняющие определенные задачи внутри системы. Понимание кода, в том числе структуры файлов и папок, принципиально важно для понимания работы WordPress в целом.

После прочтения этой главы вы будете знать, как загрузить WordPress и как пользоваться его файловой системой, а также сможете работать с ключевыми конфигурационными файлами WordPress, в том числе `wp-config.php` и `.htaccess`. В этой главе также разбираются некоторые продвинутые варианты настройки, доступные в WordPress.

Загрузка

Первым шагом на пути установки WordPress на хостинге является загрузка исходных файлов, необходимых для работы WordPress. В этом разделе вы подробнее познакомитесь с ядром системы.

Откуда загрузить

Вы можете скачать последнюю стабильную версию WordPress прямо с сайта WordPress.org, зайдя на страницу загрузок http://wordpress.org/download/.

Вы также можете обновить WordPress непосредственно из текущей копии WordPress, зайдя в раздел Обновления (Updates), который находится в **Консоли ▸ Обновления**. Щелкните на кнопке **Скачать (Download)**, чтобы загрузить последнюю версию WordPress на ваш компьютер.

WordPress также поддерживает доступ Subversion (SVN). Subversion — бесплатная система управления версиями с открытым кодом. WordPress использует Subversion для управления файлами и директориями, а также вносимыми в них изменениями. Вы можете скачать последнюю версию исходного кода WordPress, зайдя на http://core.svn.wordpress.org/trunk/.

В коренной директории SVN находится активно разрабатываемая **новейшая** версия WordPress. Обычно такая версия содержит недоработки и используется для тестирования. Запускать серьезный сайт на ней не рекомендуется.

SVN является механизмом, который разработчики активно используют для развития программного обеспечения ядра WordPress. С SVN вы можете создавать и отправлять файлы патчей («заплаток») для включения в ядро WordPress.

Доступные форматы

По умолчанию программное обеспечение WordPress скачивается в виде архива zip с названием `latest.zip`. Вы также можете скачать WordPress в виде архива tar с названием `latest.tar.gz`. Отличается только метод сжатия, файлы в архивах одинаковые.

Вы можете скачать архивы zip и tar непосредственно по ссылкам:

- http://wordpress.org/latest.zip
- http://wordpress.org/latest.tar.gz

Эти ссылки для скачивания никогда не меняются. Каждая новая версия WordPress автоматически сжимается и сохраняется в этом месте после снабжения меткой. После сохранения архива на вашем компьютере следует переименовать файл таким образом, чтобы он включал в себя номер версии WordPress, например `wordpress-3.5.zip`. Это поможет не забыть, какая именно версия WordPress была сохранена на компьютере.

Архив версий

На WordPress.org есть Архив версий WordPress. Он снабжен списком доступных для загрузки версий WordPress начиная с версии 0.71. Архив располагается по ссылке http://wordpress.org/download/release-archive/.

Не забывайте, что активно поддерживается только самая актуальная версия WordPress, поэтому версии, которые можно скачать по ссылке, существуют скорее для того, чтобы с ними сверяться, чем для актуального использования. Словосочетание «активно поддерживается» означает, что исправление ошибок, критичных

для безопасности или проблем с надежностью, делается для активной ветви и не может ретроспективно применяться для предыдущих версий. Если вам нужны исправления, придется обновить установленную версию WordPress.

Более старые версии WordPress особенно полезны для возвращения сайта к предыдущей версии. Например, если вы обновили старую версию WordPress до последней стабильной версии и столкнулись с проблемами, то сможете легко скачать версию, на которой изначально работал сайт, чтобы вернуться к ней. Архив версий также поддерживает скачивание каждой бета-версии и кандидата в окончательные версии WordPress. Приятно видеть, как WordPress получает все большее и большее распространение в качестве платформы программного обеспечения.

Архив версий также может пригодиться, если вы хотите обновить старую версию WordPress, в ядро которой были внесены изменения. Просто сравните исходный код сайта на WordPress с аналогичной версией WordPress из архива, и вы сразу увидите все отличия или изменения в коде.

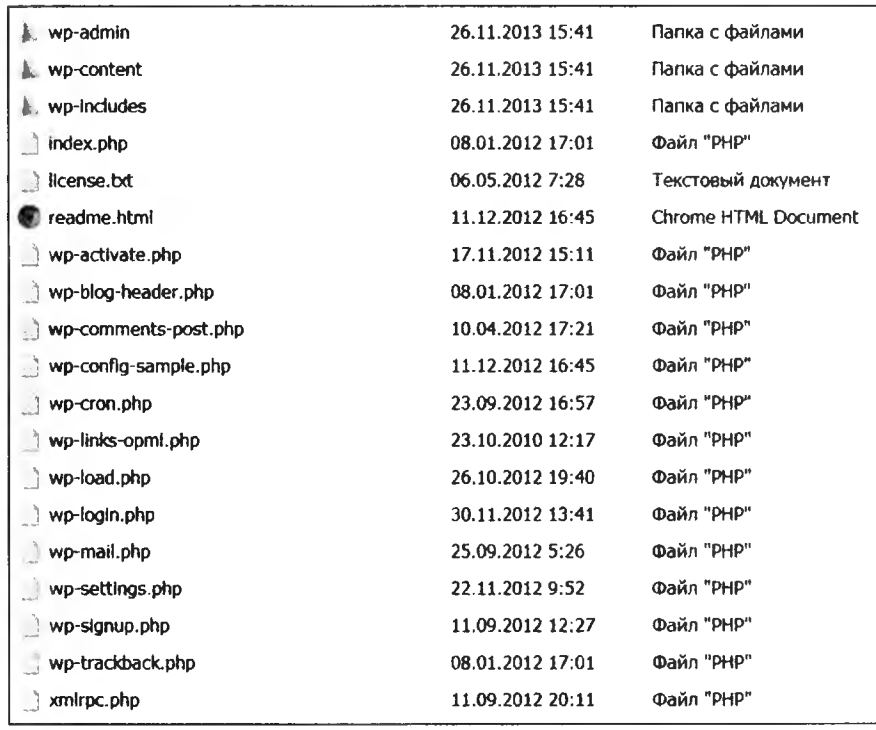
Структура папок и файлов

Исходный код WordPress включает в себя множество различных файлов на PHP, JavaScript и CSS. Каждый файл служит для своей особой цели. Красота программного обеспечения с открытыми исходниками в том, что весь код доступен, то есть вы можете с легкостью изучить его, чтобы лучше понять, как работает WordPress. Самый подходящий источник для изучения WordPress — сам WordPress.

После распаковки загруженного WordPress вы увидите набор файловых структур WordPress, как это показано на рис. 2.1.

WordPress по умолчанию содержит три директории: `wp-admin`, `wp-content` и `wp-includes`. К файлам ядра относятся все файлы из директорий `wp-admin` и `wp-includes`, а также большинство файлов из корневой директории WordPress. Директория `wp-content` содержит ваши произвольные файлы, включая темы, плагины и медиа. В ней расположен код, который контролирует управление контентом и его представлением в WordPress. HTML-содержимое WordPress, например страницы и записи, хранится в базе данных MySQL наряду с метаданными, такими как структуры тегов и категорий, обе из которых подробно рассматриваются в главе 6.

Изменение любого из файлов ядра WordPress может привести к нестабильности сайта. Например, безвредные, но некорректно внесенные изменения в функции Консоли или авторизации могут оставить вас с копией WordPress, которой невозможно управлять. Изменения в ядре также в высшей степени затрудняют обновление WordPress, поскольку все они перезаписываются при обновлении до последней версии. Как уже говорилось в предыдущем разделе, исправления критических ошибок в ядре WordPress делаются только для актуальной ветки, поэтому если вы будете вынуждены обновить WordPress по соображениям безопасности, вам придется по новой интегрировать все изменения, внесенные вами в ядро, и надеяться, что они не будут конфликтовать с другими изменениями. Поддерживать целост-



wp-admin	26.11.2013 15:41	Папка с файлами
wp-content	26.11.2013 15:41	Папка с файлами
wp-includes	26.11.2013 15:41	Папка с файлами
index.php	08.01.2012 17:01	Файл "PHP"
license.txt	06.05.2012 7:28	Текстовый документ
readme.html	11.12.2012 16:45	Chrome HTML Document
wp-activate.php	17.11.2012 15:11	Файл "PHP"
wp-blog-header.php	08.01.2012 17:01	Файл "PHP"
wp-comments-post.php	10.04.2012 17:21	Файл "PHP"
wp-config-sample.php	11.12.2012 16:45	Файл "PHP"
wp-cron.php	23.09.2012 16:57	Файл "PHP"
wp-links-opml.php	23.10.2010 12:17	Файл "PHP"
wp-load.php	26.10.2012 19:40	Файл "PHP"
wp-login.php	30.11.2012 13:41	Файл "PHP"
wp-mail.php	25.09.2012 5:26	Файл "PHP"
wp-settings.php	22.11.2012 9:52	Файл "PHP"
wp-signup.php	11.09.2012 12:27	Файл "PHP"
wp-trackback.php	08.01.2012 17:01	Файл "PHP"
xmlrpc.php	11.09.2012 20:11	Файл "PHP"

Рис. 2.1. Структура файлов и папок WordPress по умолчанию

ность и стабильность вашей копии WordPress на протяжении длительного времени гораздо проще, если вы не вносите изменения в файлы ядра.

Обычно файлы из корневой директории WordPress, а также из директорий `wp-admin`, `wp-includes` **никогда** не следует редактировать, однако в следующем разделе рассказывается о некоторых файлах корневой директории, которые могут быть изменены в рамках тонкой настройки. В целом же следуйте правилу, которому посвящена глава 4: не взламывайте ядро!

Настройка WordPress

WordPress включает в себя некоторые файлы, которые могут редактироваться в различных целях. Эти файлы могут изменяться как функции WordPress. Всегда проверяйте внесенные изменения в среде разработки, перед тем как публиковать их на рабочем сервере.

Данный раздел охватывает функции соединения с базой данных, запоминания информации FTP, активации инструментария для отладки и прочие с использованием `wp-config.php`. Здесь также рассказывается о значимости файла `.htaccess`, включая управление ограничениями памяти PHP и максимальными размерами загружаемой информации, созданием переадресации и установкой ограничений доступа.

Файл wp-config.php

Самый важный файл в любой копии WordPress — это `wp-config.php`. Он содержит все настройки соединения с базой данных, включая имя базы данных, имя пользователя и пароль для доступа к базе данных MySQL. В нем также находится информация о дополнительной базе данных и другие тонкие настройки. Файл `wp-config.php` изначально назывался `wp-config-sample.php`. Его переименование в `wp-config.php` — один из первых шагов на пути установки WordPress.

Файл `wp-config.php` обычно хранится в корневой директории WordPress. Вы также можете переместить файл `wp-config` в родительскую директорию. Так что если ваша директория WordPress расположена здесь:

```
/public_html/my_website/wp-config.php
```

вы можете спокойно переместить файл сюда:

```
/public_html/wp-config.php
```

WordPress сначала ищет файл `wp-config` в корневой директории, а потом, если не может найти его там, — в родительской. Это происходит автоматически, поэтому никаких изменений вносить не нужно.

ЗАМЕЧАНИЕ

Убрать файл `wp-config.php` из корневой директории WordPress — хорошая мера безопасности, делающая практически невозможным потенциальный доступ к этому файлу из браузера.

Некоторые параметры в WordPress хранятся как константы, их можно увидеть в файле `wp-config.php`. У всех констант одинаковый формат:

```
define( 'OPTION_NAME', 'value' );
```

`OPTION_NAME` — имя параметра, определенного постоянно; `value` — значение параметра, которое может быть обновлено до любого желательного вам значения и сохранено. При добавлении новых параметров в файл `wp-config.php` важно помещать их выше следующей строки:

```
/* Это всё, дальше не редактируем. Успехов! */
(/* That's all, stop editing! Happy blogging. */)
```

Если у вашей копии WordPress проблемы с подключением к базе данных, сюда стоит заглянуть, пытаясь решить проблему. Если вы получаете сообщение «Ошибка соединения с базой данных» («Error establishing a database connection»), в первую очередь нужно проверить, чтобы параметры `DB_NAME`, `DB_USER` и `DB_PASSWORD` были выставлены в соответствии с сервером вашей базы данных. Также убедитесь, что имя `DB_HOST` соответствует хосту вашего сервера. Обычно там стоит `localhost`, но некоторые хостинговые компании настраивают пакеты WordPress с веб-серверами и серверами MySQL на различных машинах, что приводит к необходимости специфической для данной хостинговой компании настройки расположения базы данных MySQL.

Свяжитесь с технической поддержкой вашего хостинга или обратитесь к их онлайн-документации, чтобы выставить верные значения хоста.

Вы можете изменить кодировку базы данных (`charset`), изменив значение параметра `DB_CHARSET`. По умолчанию выставлено `utf8` (Unicode UTF-8). Эта кодировка поддерживает все языки и практически всегда является наилучшим вариантом.

Начиная с версии WordPress 2.2 параметр `DB_COLLATE` позволяет определять сортировку базы данных, то есть порядок сортировки кодировки. (**Кодировка** — это набор символов, представляющих слова в языке. Сортировка определяет порядок использования при сортировке кодировки, как правило — алфавитный.) Этот параметр по умолчанию пустой и обычно должен оставаться таковым. Если вы хотите изменить сортировку базы данных, добавьте значение, подходящее для вашего языка. Этот параметр следует менять до установки WordPress. Смена данного значения после установки может вызвать проблемы в WordPress.

Безопасность WordPress можно усилить, установив в файле `wp-config.php` секретные ключи. Ключ безопасности — соль для хэширования, усложняющая взлом вашего сайта посредством добавления случайных элементов (соли) к установленному вами паролю. Эти ключи не требуются для функционирования WordPress, но они создают дополнительный слой безопасности на сайте.

Для автоматического генерирования ключей безопасности посетите ссылку для генерирования ключей для файла `wp-config.php` на WordPress.org (**https://api.wordpress.org/secret-key/1.1/salt/**), как это показано на рис. 2.2. Или же вы можете напечатать ряд случайно выбранных символов вместо текста «впишите сюда уникальную фразу» («put your unique phrase here»). Цель — использовать секретные ключи, уникальные и на 100 % случайно сгенерированные.

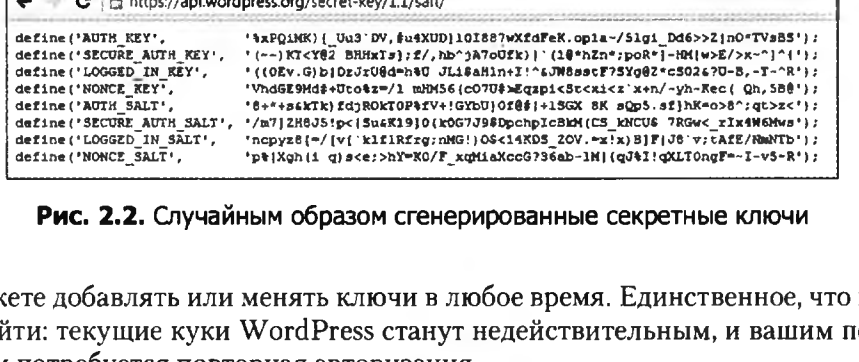


Рис. 2.2. Случайным образом сгенерированные секретные ключи

Вы можете добавлять или менять ключи в любое время. Единственное, что может произойти: текущие куки WordPress станут недействительным, и вашим пользователям потребуется повторная авторизация.

Другой особенностью обеспечения безопасности, включаемой в `wp-config.php`, является способность определять префикс таблицы базы данных для WordPress. По умолчанию этот параметр выставляется на `wp_`. Вы можете изменить это значение, присвоив переменной `$table_prefix` любое значение, как, например:

```
$table_prefix = 'bwar_';
```

Если хакер хочет взломать ваш сайт, используя SQL-инъекцию, ему будет куда сложнее выяснить имена таблиц, и, что вполне возможно, вообще отвратит его от идеи делать SQL-инъекцию. Выбор уникального префикса таблицы также делает возможным запуск нескольких копий WordPress в одной базе данных. Если вы хотите изменить префикс таблицы после установки WordPress, вы можете использовать плагин WP Security Scan (http://wordpress.org/extend/plugins/wp-security-scan/). Убедитесь, однако, что перед этим вы не забыли сделать резервное копирование.

Файл `wp-config` также содержит параметры локализации вашей копии WordPress. У WordPress есть встроенная способность для использования на различных языках. Значение переменной `WPLANG` определяет язык WordPress по умолчанию. Чтобы этот параметр работал, в `wp-content/languages` должен быть установлен соответствующий файл MO (машинный объект). Файлы MO представляют собой сжатые файлы PO (портативный объект), которые содержат перевод сообщений WordPress и текстовые строки на том или ином языке. Файлы MO и PO являются компонентами подсистемы GNU «gettext», которая лежит в основе мультиязычности WordPress. Чтобы получить полный список доступных языков MO, обратитесь к следующим ресурсам:

- **Страница Кодекса WordPress на вашем языке (in Your Language)** — http://codex.wordpress.org/WordPress_in_Your_Language
- **Репозиторий языковых файлов WordPress (Language File Repository)** — http://svn.automattic.com/wordpress-i18n/

Отладку ошибок в WordPress можно сделать проще, используя параметр `WP_DEBUG`. Будучи включенным, `WP_DEBUG` отображает ошибки WordPress на экране, вместо того чтобы замещать их белым экраном. Чтобы включить `WP_DEBUG`, установите его значение на `true`:

```
define( 'WP_DEBUG', true );
```

В новых копиях WordPress этот параметр в `wp-config` будет определен как `false`. Если этот параметр не определен, по умолчанию ему присваивается значение `false` и сообщения об ошибках не отображаются. Не забудьте отключить или удалить этот параметр, когда вы окончите отладку, поскольку сообщения об ошибках могут помочь хакерам в поиске уязвимостей вашего сайта. При разработке в WordPress параметр `WP_DEBUG` лучше держать включенным, чтобы сверяться с любыми предупреждениями и ошибками, которые могут отображаться.

Расширенные параметры wp-config

В файле `wp-config` можно работать и с расширенными параметрами. Они не включены в файл по умолчанию, их придется добавить вручную.

Чтобы указать адрес WordPress и адрес блога, используйте один из двух параметров:

```
define( 'WP_SITEURL', 'http://example.com/wordpress' );
define( 'WP_HOME', 'http://example.com/wordpress' );
```

Параметр `WP_SITEURL` позволяет временно изменить URL сайта на WordPress. Он не изменяет значение параметра базы данных для `siteurl`, а вместо этого временно меняет значение. В отсутствие этого параметра WordPress возвращается к использованию настройки `siteurl` базы данных. Параметр `WP_HOME` работает точно таким же образом, позволяя временно изменить домашнее значение для WordPress. Оба параметра требуют ввода полного URL, включая `http://`.

ЗАМЕЧАНИЕ

Это полезная технология, если вы создаете сайт на WordPress с использованием временного URL, такого как new.example.com. Вы можете просто удалить эти два параметра при переходе к полноценной работе, и WordPress загрузит рабочий URL.

В версии 2.6 был введен параметр, который позволяет менять расположение директории `wp-content`. Два необходимых для этого параметра:

```
define( 'WP_CONTENT_DIR', $_SERVER['DOCUMENT_ROOT'] .
          '/wordpress/blog/wp-content' );
define( 'WP_CONTENT_URL', 'http://domain.com/wordpress/blog/wp-content');
```

Значение параметра `WP_CONTENT_DIR` представляет собой полный локальный путь к директории `wp-content`. `WP_CONTENT_URL` — это полный URI этой директории. Дополнительно можно указать путь к директории плагинов:

```
define( 'WP_PLUGIN_DIR', $_SERVER['DOCUMENT_ROOT'] . '/blog/wp-content/plugins' );
define( 'WP_PLUGIN_URL', 'http://example/blog/wp-content/plugins');
```

`WP_PLUGIN_DIR` и `WP_PLUGIN_URL` — параметры, используемые разработчиками плагинов для определения места нахождения папки с плагинами. Если разработчик плагинов не пользуется этими константами, есть неплохой шанс, что плагин сломается в случае смены расположения директории `wp-content`. Никогда не перемещайте директорию `wp-content` на рабочем сервере без предварительного тестирования в среде разработки.

WordPress сохраняет редакции записей для каждой редактуры записи или страницы. Редакции сохраняются при нажатии на кнопку **Сохранить (Save)** или на кнопку **Опубликовать (Publish)** а также с помощью встроенной в WordPress функции автосохранения. Представьте, что у каждой вашей записи есть 10 редакций. Если у вас 100 записей, это будет 1.000 записей в базе данных. Таким образом, размер базы данных может быстро увеличиться, что способно привести к замедлению работы сайта, поскольку на выборку записей из таблицы в большой базе данных уходит больше времени. К счастью, у WordPress есть встроенный параметр для исправлений под названием `WP_POST_REVISIONS`. Вы можете присвоить ему значение `false`, чтобы полностью отключить хранение редакций, или же указать максимальное число редакций, сохраняемых для каждой записи или страницы. Вот примеры обоих сценариев:

```
define( 'WP_POST_REVISIONS', false );
define( 'WP_POST_REVISIONS', 5 );
```

Вы также можете настроить интервал автосохранения, используя параметр AUTOSAVE_INTERVAL. При редактировании записи для автосохранения изменений WordPress использует AJAX. По умолчанию интервал составляет 60 секунд. Интервал автосохранения в секундах вы можете установить в wp-config. Установите 5 минут, используя код:

```
define( 'AUTOSAVE_INTERVAL', 300 );
```

Прекрасным отладочным параметром является SAVEQUERIES. Активация этого параметра сохраняет все запросы базы данных в глобальный массив, который может отображаться на вашей странице. Это может помочь в отладке проблем с запросами, а также позволит увидеть, что именно делает WordPress при загрузке каждой страницы. Если вы работаете над темой или плагином и не можете получить отображение нужного числа записей, этот отладочный параметр покажет, что именно WordPress запрашивает у базы данных. Активируйте его, присвоив ему значение true:

```
define( 'SAVEQUERIES', true );
```

Чтобы отобразить массив запросов в вашей теме, добавьте следующий код в любой файл шаблона темы:

```
if ( current_user_can( 'manage_options' ) ) {
   global $wpdb;
   print_r( $wpdb->queries );
}
```

Предшествующий код отображает сохраненный массив запросов, если у авторизованного пользователя есть возможность управлять параметрами. Важно ограничивать все таким образом, чтобы только администраторы видели информацию на выходе. Темы и файлы шаблонов рассматриваются в главе 9 «Разработка тем».

Вы также можете активировать ведение журнала регистрации (логов) непосредственно из файла wp-config. Чтобы сделать это, сначала необходимо создать файл php_error.log и загрузить его в корневую директорию WordPress. Затем просто активируйте параметр PHP log_errors PHP и укажите путь к файлу с журналом:

```
@ini_set( 'log_errors','On' );
@ini_set( 'display_errors','Off' );
@ini_set( 'error_log','/public_html/wordpress/php_error.log' );
```

Теперь все ошибки будут сохраняться в этом файле, в том числе и все ошибки, которые выдает активация параметра WP_DEBUG, рассмотренного ранее. В предыдущем примере параметр display_errors имеет значение Off, что прекрасно для работающего сайта, поскольку вы вряд ли захотите, чтобы сообщения об ошибках отображались. Если вы занимаетесь отладкой и хотите видеть ошибки в реальном времени, просто установите этот параметр на On. Помните, что значение error_log относится к корневой директории документации веб-сервера, а не к корневой директории WordPress.

Вы также можете установить предел памяти, которую может использовать WordPress, с помощью параметра WP_MEMORY_LIMIT. Если ваш сайт превысит

допустимую для работы WordPress границу, вы увидите ошибку «Допустимый размер памяти в xxxxx байт израсходован» («Allowed memory size of xxxxx bytes exhausted»). Увеличение допустимого объема памяти решает эту проблему. Память ограничивается путем указания нужного количества мегабайт:

```
define( 'WP_MEMORY_LIMIT', '32M' );
```

Этот параметр функционирует, только если он разрешен вашей хостинговой компанией. Некоторые хостинговые компании не позволяют динамически менять объем памяти и устанавливают очень низкое значение. С такой проблемой обычно сталкиваются при использовании недорогих хостингов, которые держат цены на таком уровне, помещая несколько копий веб-сервера на один физический хост, что приводит к проблемам с дисковым пространством.

Таким образом вы увеличиваете предел памяти только для WordPress, а не для других приложений, запущенных на сервере. Чтобы увеличить предел для всех сайтов, установите значение параметра `php_value memory_limit` в файле `php.ini`. Например, при импортировании большого количества контента, скажем, записи блога за месяцы или за годы, вы, вероятнее всего, превысите предел памяти.

Одной из замечательных функций WordPress является встроенная локализация. По умолчанию WordPress отображается на английском, но легко может быть перенастроен на любой другой язык, на который он переведен. Установка параметра `WPLANG` заставит WordPress загрузить нужные языковые файлы:

```
define ( 'WPLANG', 'en-GB' );
```

Значение параметра, показанное ранее, состоит из кода языка ISO-639, за которым следует код страны ISO-3166. Таким образом, `en-GB` — это английский (Великобритания). Эта настройка отсылает вас к файлам `.mo` и `.po` для перевода языка.

Вы также можете определить параметр `LANGDIR`. Этот параметр отвечает за то, в какой директории будут храниться языковые файлы `.mo`. По умолчанию файлы `.mo` WordPress ищет в `wp-content/languages`. Если вы хотите переместить эту папку, просто установите параметр `LANGDIR`:

```
define( 'LANGDIR', '/wp-content/bury/my/languages' );
```

Теперь WordPress будет искать фалы `.mo` в новом месте.

`CUSTOM_USER_TABLE` и `CUSTOM_USER_META_TABLE` — существенные параметры. Они пригодятся, если вы хотите иметь две или более отдельные копии WordPress с одной и той же учетной записью пользователя. Не забудьте определить их до установки WordPress.

```
define( 'CUSTOM_USER_TABLE', 'joined_users' );
define( 'CUSTOM_USER_META_TABLE', 'joined_usermeta' );
```

Активация этих двух параметров позволяет вам определить имя пользователя WordPress по умолчанию и пользователя метатаблицы. Это действие приводит к тому, что оба сайта получают общую информацию, включая имена пользователей,

пароли, биографию автора и т. д. Это прекрасный способ установить новую копию WordPress, не теряя синхронизации с текущей пользовательской базой.

Если вы хотите, чтобы у ваших пользователей были разные роли в рамках одной копии WordPress, но по-прежнему общая учетная запись, не используйте параметр `CUSTOM_USER_META_TABLE`. Все, что хранится в таблицах пользователя, останется на месте, но остальные элементы будут определяться блогом (то есть уровень пользователя, имя и фамилия и т. д.).

Вы можете использовать многочисленные параметры куки, такие как `COOKIE_DOMAIN`, `COOKIEPATH` и `SITECOOKIEPATH`. Эти параметры обычно используются в версии WordPress Multisite, применяющей для сайтов поддомены. Они позволяют установить первичный домен, чтобы куки могли создаваться и подтверждаться для всех поддоменов в сети:

```
define( 'COOKIE_DOMAIN', '.domain.com' );
define( 'COOKIEPATH', '/' );
define( 'SITECOOKIEPATH', '/' );
```

Обычно у вас нет необходимости использовать или менять этот параметр, но если начались проблемы с куки, посмотреть в первую очередь следует именно сюда.

С момента появления автоматической установки плагинов и тем, а также автоматизации процесса обновления настройки FTP можно прописывать прямо в файле `wp-config`. Это необходимо, только если ваш хост не поддерживает процесс автоматической установки, что несложно определить, если каждый раз при попытке установки плагина или темы WordPress запрашивает у вас информацию об FTP.

Чтобы сохранить информацию об FTP в WordPress, добавьте в файл `wp-config` следующие параметры:

```
define( 'FTP_USER', 'username' );
define( 'FTP_PASS', 'password' );
define( 'FTP_HOST', 'ftp.example.com:21' );
```

Просто введите имя пользователя для FTP, пароль и хост с портом — и дело сделано! WordPress больше не будет просить у вас информацию об FTP при использовании автоматической установки.

Вы можете добавить дополнительные параметры FTP/SSH для различных настроек:

```
// метод обращения к файловой системе: "direct", "ssh", "ftpext", или "ftpsockets"
define( 'FS_METHOD', 'ftpext' );
// абсолютный путь к корневой директории, где установлен WordPressabsolute path to
root installation directory
define( 'FTP_BASE', '/public_html/wordpress/' );
// абсолютный путь к директориии wp-content
define( 'FTP_CONTENT_DIR', '/public_html/wordpress/wp-content/' );
// абсолютный путь к директории wp-plugins
define( 'FTP_PLUGIN_DIR ', '/ public_html /wordpress/wp-content/plugins/' );
// абсолютный путь к публичным ключам SSH
define( 'FTP_PUBKEY', '/home/username/.ssh/id_rsa.pub' );
```

```
// абсолютный путь к персональным ключам SSH
define( 'FTP_PRIVKEY', '/home/username/.ssh/id_rsa' );
// безопасное SSL-соединение к FTP, если поддерживается хостинговой компанией
define( 'FTP_SSL', false );
```

Вы также можете перезаписать разрешения для файлов, установленные в WordPress по умолчанию, используя параметры `FS_CHMOD_FILE` и `FS_CHMOD_DIR`:

```
define( 'FS_CHMOD_FILE',0644 );
define( 'FS_CHMOD_DIR',0755 );
```

Значения в виде однозначных чисел представляют собой разрешения для типов Пользователь (User), Группа (Group) и Мир (World), установленные для файлов и папок на вашем веб-сервере. Чтобы узнать больше о WordPress и разрешениях для файла, зайдите по ссылке http://codex.wordpress.org/Changing_File_Permissions.

Эти параметры могут помочь при работе с некоторыми хостинговыми компаниями, которые практикуют ограничение разрешений на работу с файлами для всех пользователей. Настройки сервера будут перезаписаны, что позволит WordPress обновляться автоматически.

Параметр `WP_CACHE` требуется для работы некоторых кэш-плагинов. Активация этого параметра будет включать файл `wp-content/advanced-cache.php`. Чтобы активировать его, используйте следующий код:

```
define( 'WP_CACHE', true );
```

У WordPress есть многочисленные постоянные параметры, которые вы можете установить. Вот функция PHP, которая позволяет просмотреть все константы, установленные на данный момент для вашей копии:

```
print_r( @get_defined_constants() );
```

Один из более сложных параметров принуждает авторизоваться на вашем сайте WordPress. Это потребует от пользователей авторизации по ссылке HTTPS, причем все данные, предаваемые с сайта и на него, будут шифроваться. Чтобы активировать авторизацию SSL, добавьте параметр `FORCE_SSL_LOGIN` следующим образом:

```
define( 'FORCE_SSL_LOGIN', true );
```

Вы также можете заставить все администраторские страницы использовать SSL. Эта возможность активируется с помощью параметра `FORCE_SSL_ADMIN`:

```
define( 'FORCE_SSL_ADMIN', true );
```

Все администраторские страницы консоли (`/wp-admin`) теперь будут шифроваться SSL. Не забывайте, что активация этой настройки замедляет загрузку администраторской страницы, но все данные, проходящие через WordPress, будут шифроваться с использованием SSL. Также не забудьте, что ваш сайт должен быть сконфигурирован для работы с SSL. Самый простой способ проверить это — зайти на сайт с использованием https, например https://example.com. Если страница загружается, SSL на вашем сервере установлен.

Принудительный SSL для администратора WordPress — большое преимущество в безопасности. Все данные, проходящие через WordPress, будут шифроваться, что страхует от потенциальной кражи авторизационной информации WordPress.

Начиная с версии 2.9 в WordPress присутствует мусорная корзина. Она содержит все записи, страницы, приложения и комментарии, которые были удалены. Это позволяет восстановить любой контент, случайно удаленный из WordPress. По умолчанию корзина опустошается каждые 30 дней. Этот процесс безвозвратно удаляет все объекты. Вы можете задать другой интервал, изменив значение параметра `EMPTY_TRASH_DAYS`:

```
define( 'EMPTY_TRASH_DAYS', 7 );
```

Теперь корзина будет автоматически опустошаться каждые 7 дней. Вы также можете полностью отключить корзину, выставив значение параметра на 0. Теперь ссылка на корзину будет заменена на «Удалить навсегда» («Delete Permanently»). Не забывайте, что при щелчке на ней WordPress не будет требовать подтверждения.

Есть также параметр для деактивации WordPress cron. Cron используется в WordPress для выполнения задач по расписанию. К выполняемым по расписанию задачам общего характера относятся публикация отложенных записей и проверка наличия новых версий WordPress, тем и плагинов. Чтобы отключить WordPress cron, добавьте следующий параметр в файл `wp-config`:

```
define( 'DISABLE_WP_CRON', true );
```

В данном разделе было рассмотрено множество часто используемых параметров для `wp-config`. В WordPress есть гораздо больше менее известных параметров. Отличным ресурсом для получения информации о них является Кодекс: http://codex.wordpress.org/Editing_wp-config.php.

.htaccess

Файл `.htaccess` используется в первую очередь для создания на сайте «красивых» постоянных ссылок и URL с добавлением ключевых слов. WordPress по умолчанию создает уродливые ссылки на основе строк запросов, обычно с указанием идентификатора, как, например, http://example.com/?p=45. Такие URL полностью функциональны, но не слишком хороши для поисковых систем и посетителей сайта. Активируя «красивые» постоянные ссылки, WordPress создает URL на основе содержимого сайта, такого как названия записей и страниц, категории и метки и даты для архивов.

Активация постоянных ссылок

Чтобы активировать постоянные ссылки, зайдите в Параметры ▸ Постоянные ссылки Консоли WordPress, как это показано на рис. 2.3. Выберите любую структуру постоянных ссылок, отличную от варианта по умолчанию, и щелкните на ссылке Сохранить изменения (Save Changes).

Сохраняя изменения, WordPress пытается создать файл `.htaccess` по умолчанию. Если корневая директория WordPress открыта для записи сервером, файл создается автоматически. Если WordPress не может создать файл `.htaccess`, вы увидите инструкцию по его созданию вручную, как это показано на рис. 2.4.

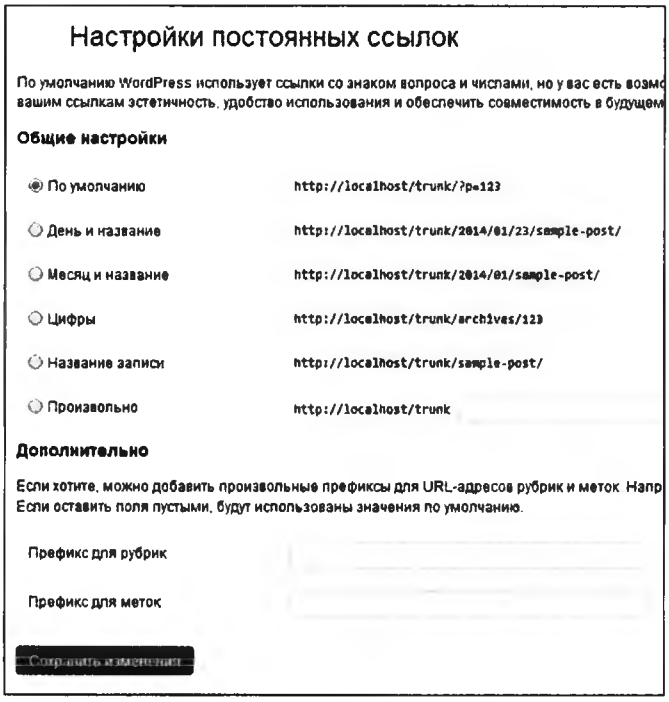


Рис. 2.3. Активация постоянных ссылок в WordPress

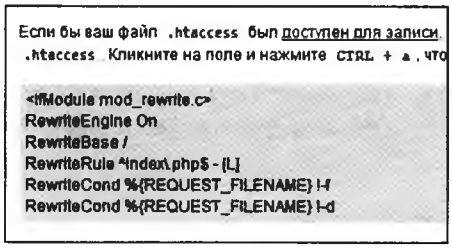


Рис. 2.4. Информация по созданию файла .htaccess вручную

Выбор структуры постоянных ссылок, основанной на месяце и годе, как эта,

```
/%year%/%monthnum%/%postname%/
```

приводит к созданию постоянной ссылки следующего вида:

```
http://example.com/2012/10/halloween-party/
```

Использование постоянных ссылок дает много преимуществ, например:

- **Оптимизация для поисковых систем (SEO).** Ключевые слова в URL могут оказать вашему сайту большую услугу с точки зрения SEO. Поисковые системы будут использовать эти ключевые слова в своих алгоритмах для позиционирования сайта в поисковых результатах.
- **Совместимость снизу вверх.** Вне зависимости от того, какую платформу использует ваш сайт (WordPress, Drupal, Joomla!), наличие надежной структуры постоянных ссылок может быть легко скопировано при «переезде».
- **Удобство в использовании.** URL, построенные с использованием идентификаторов, неудобны для пользователей. Невозможно определить содержание страницы, зная только ее идентификатор.
- **Возможность поделиться.** В интернет-эру социальных сетей делиться информацией — естественное выражение виртуального присутствия. Ключевые слова в URL сделают вашу ссылку удобной для нахождения и снабдят ее контекстом.

Правила преобразования .htaccess

Обычно веб-сервер берет URL, который относится к файлу в файловой системе документации сервера, загружает файл и обрабатывает его содержание, чтобы сгенерировать HTML, отправляемый обратно в браузер пользователя. Для файлов WordPress, таких как `wp-login.php`, это точное описание генерирования экрана авторизации. При получении «красивой» ссылки, такой как `example.com/2012/travel/haddonfield`, веб-серверу нужно просто запустить основной цикл WordPress, чтобы код ядра мог проанализировать URL и превратить его в запрос для базы данных, который найдет запись с заголовком Haddonfield в категории Travel. В отличие от статического сайта, для которого вам понадобилось бы создавать файл с этим именем, WordPress хранит его содержимое в базе данных. Напрямую загружаются всего несколько файлов.

«Секретный ингредиент» механизма постоянных ссылок WordPress суммируется в трех правилах преобразования, добавляемых в файл `.htaccess` при активации постоянных ссылок:

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.php [ L]
```

Все довольно просто. Эти правила проверяют ссылки URL, используемые для доступа к сайту, чтобы увидеть, относятся ли они к существующему файлу или к директории в иерархии файловой системы. Нотации `!-f` и `!-d` являются отрицаниями; `.htaccess` убеждается, что URL не соотносится с каким-либо действительным путем к файлу или директории. Если URL на самом деле соответствует имеющемуся файлу, например административной функции WordPress, такой как `wp-login.php`, то преобразования не делается и веб-сервер пробует загрузить этот файл (чтобы

выполнить содержащийся в нем код PHP). Если по пути, определенному URL, файла или директории не существует, то входящая ссылка преобразуется в `index.php`, запуская ядро системы WordPress. Вы изучите шаги, необходимые для конвертирования строки URL в запрос MySQL, более подробно в предисловии к дискуссии о цикле отображения контента в главе 5.

ЗАМЕЧАНИЕ

Простая проверка на существование файла или директории может иметь неожиданные побочные эффекты, если контент, не относящийся к WordPress, помещен в ту же самую структуру директории, что и код WordPress. Например, директория с изображениями, как равноправная `wp-content`: `example.com/wp-content` и `example.com/images`. Вы можете решить обойти медиабиблиотеку WordPress, если эти изображения управляются своим собственным набором механизмов. Что происходит, если пользователь создает URL с неверно набранным именем, указывающим на несуществующий файл? Правило преобразования `.htaccess` не срабатывает, поскольку файла с таким именем не существует, и запускается ядро WordPress. Ожидающий увидеть изображение пользователь вместо этого получает контент с ошибкой 404 для несуществующего URL. Если вы собираетесь добавлять директории рядом с копией WordPress, либо поместите WordPress в отдельную поддиректорию (`example.com/wordpress`), либо добавьте в `.htaccess` правило преобразования, которое будет распознавать добавленные равноправные директории и немедленно перенаправлять эти URL на веб-сервер:

```
RewriteRule ^images/(.*) images/$1 [L]
```

По сути, правило гласит: «Возьми любой URL, начинающийся с компонента `images`, и передай его веб-серверу». Директива [L] означает: «Остановить обработку после достижения соответствия правилу», — а само преобразование просто передает назад то, что было получено. Если у вас несколько директорий, расположенных параллельно с копией WordPress, вам понадобится по одному правилу преобразования для каждой.

Файл `.htaccess` также может управлять перенаправлением URL. Если вы меняете адрес страницы «Обо мне» («About») с http://example.com/about на http://example.com/about-me, любой, кто зайдет по изначальной ссылке, получит «страницу 404». Перенаправление URL отправляет со старой ссылки на новую, чтобы посетители не заблудились. Также произойдет уведомление о новом URL поисковых систем, чтобы они смогли обновить индекс.

Ниже приведен пример постоянного перенаправления 301 для статической страницы:

```
redirect 301 /about http://example.com/about-me
```

WordPress проводит некоторые дополнительные преобразования и «чистит» URL для улучшения результатов в поисковых системах, как вы увидите в главе 5.

Настройка управления через .htaccess

Файл `.htaccess` — мощная штука, которая может контролировать гораздо больше, чем просто структуру URL. Например, используя `.htaccess`, вы можете управлять

настройками PHP. Чтобы увеличить объем памяти, выделенный для PHP, используйте команду:

```
php_value memory_limit 64M
```

Таким образом предел памяти в PHP будет увеличен до 64 Мбайт. Вы также можете увеличить максимальный размер загружаемого файла и размер записи:

```
php_value upload_max_filesize 20M
php_value post_max_size 20M
```

Теперь максимальный размер файла, который вы можете опубликовать через форму и загрузить, составляет 20 Мбайт. Большинство хостинговых компаний по умолчанию выставляют эти значения на уровне примерно 2 Мбайт, так что эти настройки часто используются для загрузки файлов большего размера. Не все хостинговые компании позволяют выставлять эти значения в файле `.htaccess`, так как в таком случае они могут привести к ошибкам на вашем сайте.

Файл `.htaccess` также может использоваться в целях обеспечения безопасности. Использование `.htaccess` позволяет вам запретить доступ к сайту по IP-адресу, по сути, закрывая его от любых анонимных посетителей. Чтобы закрыть сайт по IP-адресу, добавьте в файл `.htaccess` следующий код:

```
AuthUserFile /dev/null
AuthGroupFile /dev/null
AuthName "Access Control"
AuthType Basic
order deny,allow
deny from all
#IP address to whitelist
allow from xxx.xxx.xxx.xxx
```

Замените `xxx.xxx.xxx.xxx` на любой IP-адрес, которому вы хотите открыть доступ к вашему сайту. Вы можете сделать много строк `allow from` и, таким образом, добавить столько IP-адресов, сколько нужно. Теперь доступ к вашему сайту будет возможен только при использовании одного из IP-адресов, определенных здесь.

Более широко этот параметр используется для блокировки директории `wp-admin`. Он делает возможным доступ к URL административной консоли только с определенных IP-адресов, что сильно усложнит задачу взлома WordPress через сервер базы данных. Чтобы активировать блокировку, создайте отдельный файл `.htaccess` с указанным выше кодом в директории `wp-admin`.

Не забывайте, что большинство интернет-провайдеров присваивают клиентам адреса динамически, поэтому IP-адрес используемого вами компьютера будет время от времени меняться. Если вы оказались заблокированы, просто обновите файл `.htaccess`, введя новый IP-адрес, или же удалите файл целиком. Это не очень хорошо в случае, если на вашем сайте есть открытая регистрация, поскольку у пользователей должен быть доступ к директории `wp-admin`.

Вы также можете регулировать доступ с помощью IP-адресов, выраженных групповыми символами. Например, `123.123.123.*` откроет доступ любому, у кого первые три октета IP-адреса совпадают с указанными, при этом четвертый выражен групповым символом. Например, `123.123.123.110-230` откроет доступ любому с IP-адресом в диапазоне от `123.123.123.110` до `123.123.123.230`.

Из файла `.htaccess` можно активировать и ведение журнала ошибок. Первый шаг — создать файл `php-errors.log` в корневой директории WordPress. Затем добавьте в файл `.htaccess` следующий код, чтобы активировать ведение журнала ошибок:

```
php_flag display_startup_errors off
php_flag display_errors off
php_flag html_errors off
php_flag log_errors on
php_value error_log /public_html/php-errors.log
```

Таким образом активируется ведение журнала, но подавляется отображение каких-либо сообщений об ошибках. Это прекрасная установка для рабочей среды, поскольку не нужно, чтобы ошибки отображались публично.

Файл .maintenance

У WordPress есть встроенный режим технического обслуживания, который может быть активирован с помощью файла `.maintenance`. Файл `.maintenance` используется WordPress во время процесса автоматического обновления. Это оберегает посетителей от наблюдения каких-либо сообщений об ошибках при обновлении файлов ядра WordPress. Чтобы опробовать эту функцию, просто создайте новый файл `.maintenance` и добавьте в него следующую строку кода:

```
<?php $upgrading = time(); ?>
```

Добавьте этот файл в корневую директорию WordPress, и ваш сайт сразу же перейдет в режим технического обслуживания. Он будет закрыт для всех посетителей, вместо контента будет отображаться универсальное сообщение о техническом обслуживании: «Временно недоступен из-за плановых работ. Зайдите через пару минут» («Briefly unavailable for scheduled maintenance. Check back in a minute»). Функция `time()` может быть заменена любой временной отметкой в формате для UNIX.

Вы можете произвольно настроить эту страницу, создав файл `maintenance.php` и поместив его в директорию `wp-content`. WordPress использует этот файл для отображения во время любых установленных вами периодов технического обслуживания. Так вы можете создать индивидуальное сообщение о техническом обслуживании для посетителей вашего сайта.

Этот файл также используется при процессе автоматического обновления WordPress. Файл `.maintenance` создается непосредственно перед тем, как

WordPress устанавливает новые файлы ядра во время обновления. Это гарантирует, что во время данного процесса посетители не увидят никаких сообщений об ошибках.

Пользовательская площадка wp-content

В директории `wp-content` хранятся практически все файлы для пользовательской настройки WordPress. Здесь находятся ваши плагины, темы и дополнительные файлы для расширения WordPress в любом вообразимом направлении.

В директории `wp-content` есть только один файл PHP — `index.php`. Вот его содержимое:

```
<?php
// Silence is golden.
// Молчание - золото.
?>
```

Так зачем же он нужен? На самом деле это очень важный файл. Он не дает никому видеть список файлов в папке `wp-contents`. Если файл `index.php` не существует, а ваш веб-сервер позволяет составление списков файлов в директориях, зайдя на http://example.com/wp-contents, можно увидеть все файлы и папки в этой директории. Это может пригодиться хакерам в получении доступа к файлам ключей, что позволит взломать ваш сайт. Например, если уязвимость была выявлена в плагине, возможность увидеть список файлов в директории плагинов WordPress позволит атакующему быстро и легко получить информацию о том, является ли ваш сайт подходящей целью.

Если вы обновляете WordPress вручную, убедитесь, что избегаете перезаписывания директории `wp-content`.

Плагины

Плагины хранятся в директории `wp-content/plugins`. Плагин может представлять собой один или несколько файлов внутри папки. Любые файлы в директории `/plugins` сканируются WordPress, чтобы определить, является ли файл надлежащим образом отформатированным плагином WordPress. Если файл определяется как плагин, он появляется в консоли администратора во вкладке Плагины ▸ Установленные и готов к активации.

ЗАМЕЧАНИЕ

Помните, что для автоматической деактивации плагина достаточно просто удалить его из папки `/plugins`. Если каких-то файлов действующего плагина не хватает, WordPress деактивирует его до того, как начнет формировать ваш сайт.

В директории `wp-content` также должна находиться директория `/mu-plugins`. Обязательные к использованию плагины (must-use plugins) — это плагины, которые активируются в WordPress автоматически. Любые плагины, находящиеся в этой папке, будут выполняться как стандартно активированные плагины. Основным отличием является то, что эти плагины не могут существовать в поддиректории, тогда они игнорируются. Об этом можно больше узнать, пройдя по ссылке http://codex.wordpress.org/Must_Use_Plugins.

Мы вернемся к плагинам в главе 8 «Разработка плагинов».

Темы

Темы хранятся в директории `wp-content/themes`. Каждая тема должна существовать в собственной поддиректории и состоять из корректно оформленных файлов-шаблонов, чтобы WordPress распознал ее как тему, пригодную для использования. В директории темы должны иметься как минимум файлы `index.php` и `style.css`, а также правильная маркировка, чтобы тема отображалась в консоли администратора во вкладке **Внешний вид ▸ Темы**.

WordPress может хранить в этой директории столько тем, сколько позволяет сервер. Вы можете легко просмотреть любую имеющуюся тему или активировать новую во вкладке **Внешний вид ▸ Темы**. Более детально мы рассмотрим темы в главе 9.

Загрузки и директория медиафайлов

WordPress хранит загруженные медиафайлы в папке `wp-content/uploads`. Эта директория не существует в копии WordPress по умолчанию. Она создается при первой успешной загрузке файла в WordPress.

По умолчанию WordPress хранит загрузки в папке по месяцам и годам. Так, загруженные изображения будут храниться примерно следующим образом:

```
/wp-content/uploads/2012/06/image.png
```

Перед тем как вы сможете загружать какие-либо изображения или файлы в WordPress, необходимо разрешить запись информации в директорию `/wp-content`. При загрузке первого изображения WordPress автоматически создает директорию `/uploads` и все необходимые поддиректории. После того как вы успешно загрузили первое изображение, верните права для `/wp-content` на запрет записи, обычно 755. На данный момент нет возможности импортировать изображения, загруженные по FTP в библиотеку файлов WordPress. Если нет возможности разрешить запись для директории `uploads`, существуют плагины (такие, как NextGen Gallery, подробно описанный далее в разделе «Персональные директории»), дающие такую функциональность.

WordPress Multisite хранит загруженные медиафайлы иначе. Вместо одной директории для загрузок Multisite создает директорию `blogs.dir`. Внутри этой папки — многочисленные поддиректории, названные по идентификатору (ID).

Идентификатор представляет собой идентификатор блога, к которому относится папка. У каждого сайта в сети Multisite есть свой уникальный идентификатор блога. Этот вопрос более детально рассмотрен в главе 10. Например, первая директория для загрузок WordPress Multisite будет выглядеть так:

```
/blogs.dir/1/files/
```

Это помогает держать загрузки для каждого сайта отдельно и облегчает их обслуживание.

Директория Upgrade

Директория `wp-content/upgrade` создается WordPress автоматически при использовании процесса автоматического обновления. Эта папка используется WordPress для хранения новой версии WordPress, скачанной с WordPress.org. Сжатые файлы WordPress после скачки извлекаются в эту папку перед обновлением. Эту папку не следует трогать, чтобы процесс автоматического обновления протекал успешно. Если данная директория удалена, WordPress воссоздаст ее при следующем автоматическом обновлении.

Персональные директории

Некоторые из плагинов, которым нужно много персональных файлов, хранят их в директории в папках `wp-content`.

Плагин Super Cache (http://wordpress.org/extend/plugins/wp-super-cache/) создает директорию `/wp-content/cache` для хранения всех кэшированных страниц вашего сайта. Кэшированная страница — это просто полностью сгенерированная страница вашего сайта, сохраненная как статический файл HTML. Вместо того чтобы генерировать страницу каждый раз, когда пользователь щелкает по ссылке, кэш-плагин выдает посетителю статический файл HTML. Это значительно уменьшает время загрузки WordPress и увеличивает производительность, поскольку страницы генерируются не при каждом просмотре, но только когда кэш перезаписывается в соответствии с вашими настройками.

Плагин Super Cache также добавляет два файла в директорию `wp-content`: `advanced-cache.php` и `wp-cache-config.php`. Они необходимы для корректной работы Super Cache. Будучи активированным, данный плагин пытается создать эти два файла. Если ему это не удается, появляется соответствующее предупреждающее сообщение. Файлы находятся в директории плагина Super Cache и могут быть вручную перемещены в директорию `wp-content`.

Самый популярный плагин для галереи изображений — NextGen Gallery (http://wordpress.org/extend/plugins/nextgen-gallery/) — создает директорию `/wp-content/gallery` для хранения всех изображений, загруженных в ваши галереи NextGen. Каждая созданная галерея представляет собой поддиректорию `/gallery`. Это помогает сохранять файлы галереи изображений в порядке и легко работать с ними.

Плагин WP-DB Backup (http://wordpress.org/extend/plugins/wp-db-backup/) создает папку `/wp-content/backup-b158b` (где `b158b` — произвольная строка), чтобы хранить резервные копии вашей базы данных. Когда вы выбираете настройку **Сохранить на сервер (Save to Server)**, все файлы резервных копий базы данных сохраняются в эту директорию. Важно не удалять резервные копии, пока вы не убедитесь, что они вам больше не нужны.

Резюме

В этой главе была рассмотрена загрузка исходных файлов WordPress. Также мы описали настройку ключевых файлов ядра WordPress: `wp-config.php` и `.htaccess`, включая более тонкую настройку каждого из них. Кроме того, был приведен обзор директории `wp-content` и вопросов взаимодействия WordPress с персональными директориями.

После обзора структуры и настроек WordPress самое время научиться создавать локальную среду разработки, чтобы можно было начать пользовательскую подгонку и разработку, не затрагивая работающий сайт.

Работаем с WordPress локально

3

В этой главе:

- Разработка локально
- Начало работы с локальной средой разработки
- Настройка локальной среды разработки — советы и хитрости
- Перемещение локально разработанного проекта на рабочий сервер

Теперь, когда вы знаете, как получить WordPress и как его установить, давайте посмотрим, как начать делать в WordPress что-то поинтереснее, чем просто использовать систему в качестве движка сайта. Как вы уже поняли из главы 1, любой пользователь может установить WordPress и использовать его в качестве основы веб-сайта. Но это только одна из причин успеха WordPress.

Как разработчику вам необходима полностью оснащенная площадка в режиме песочницы (sandbox), чтобы ставить эксперименты, проверять новые идеи и выяснять, в чем именно ошибка, не выводя из строя рабочий или открытый для всех сайт. В качестве первого шага в создании чего бы то ни было с целью дальнейшего использования WordPress для ваших проектов предлагаем ознакомиться с преимуществами установки локальной среды разработки на вашем компьютере или ноутбуке. Эта глава начинается с краткого отступления от WordPress для обсуждения разработки программного обеспечения в целом.

Преимущества локальной работы

Локальная разработка считается лучшей практикой. Обычно вам не хочется вести активный процесс разработки на рабочем веб-сайте, поскольку в любой момент на сайт могут заглянуть посетители, а разработка связана с многократным запуском

неисправного кода, который нужно вновь заставить работать. Едва ли это то, что вы хотели бы показать посетителям.

Что такое «локальная разработка»? В двух словах это значит, что у вас есть копия WordPress, в которую вы можете вносить изменения, добавлять новый код и безнаказанно совершать ошибки. Это песочница — первый элемент в цикле успешного внедрения.

Типичный цикл внедрения

Перед тем как разбираться с причинами, по которым лучше заниматься разработкой локально, посмотрим, что такое различные фазы внедрения. Внедрение включает в себя перенос кода из базовых версий разработки, который вы считаете готовым для предъявления всему свету, посредством установки и тестирования на рабочем сайте. В целом есть три уровня. В некоторых случаях их может быть больше, но эти три шага — ключевые: разработка, перенос и работа. Это базовый рабочий процесс разработки программного обеспечения, который применяется куда шире, чем разработка только на WordPress.

Первичной является среда разработки, в которой вы делаете повседневную работу. Как вы увидите в этой главе, речь в основном идет о вашем компьютере (рабочей станции) или ноутбуке. Однако в некоторых случаях это может быть среда разработки на удаленном сервере. Лучше всего разрабатывать свое приложение на платформе с системой того же типа, что и рабочая среда, где оно будет использоваться. Правда, это не всегда практично. Например, ваш рабочий сервер — сервер с оборудованием профессионального класса, на котором установлен Linux, но поскольку вашим разработчикам необходим доступ к корпоративным ресурсам, таким как Microsoft Exchange, они используют для разработки рабочие станции с Windows.

Именно поэтому есть второе звено: перенос или тестовая среда. После того как разработчик протестировал свое решение в среде разработки, он готовится переместить его на вспомогательный сервер. Задача вспомогательного сервера — стать мостиком между средой разработки и конечной рабочей средой, предотвращая поломку работающего сайта. Как вы увидите далее в этой главе, есть определенные нюансы, которые необходимо принимать во внимание при разработке межплатформенного кода — это код, который должен работать на Windows, Mac OS X или Linux. Эта промежуточная среда дает разработчику возможность убедиться, что код будет работать на сервере, схожем с рабочим сервером. При разработке для WordPress такой промежуточной средой может быть скрытый тестовый сайт на рабочем сервере.

Наконец, если решение ведет себя на промежуточном сервере в соответствии с ожиданиями, оно может быть внедрено на рабочий сервер. Рабочий сервер (или серверы) — это тот сервер, который обеспечивает нахождение веб-сайта в Интернете. Используя этот трехстадийный рабочий процесс, разработчики выигрывают от разработки локально.

Почему так много этапов?

Теперь, когда вы имеете базовое представление о рабочем процессе, вы, конечно, вернетесь к началу и спросите, почему разработчику нужно делать так много дополнительных шагов на пути внедрения кода. Хотя множество этапов едва ли сочетаются с мантрой «Пусть код заработает быстро», преимущества покрывают издержки.

Во-первых, как мы объяснили ранее, разработка локально позволяет проверять и пробовать разные вещи, не ломая работающий сайт. Пожалуй, это один из самых важных аспектов данной системы. Если ваш сайт вышел за рамки «домашних радостей», вам захочется свести время его простоя к минимуму. Разработчикам никогда не следует пробовать что-то на работающем веб-сайте.

Во-вторых, преимуществом также является конфиденциальность. Разработка локально означает, что вы работаете на своем компьютере или, иногда, в локальной сети. Вы контролируете доступ к тому, что делаете. Если же вы работаете на открытом веб-сервере, хотя и существуют способы ограничить доступ, ваша потенциальная аудитория — весь мир.

Конфиденциальность дает вам возможность пробовать разные вещи и экспериментировать. Представьте себе, что это ваша личная песочница WordPress, куда никто не заглядывает. Например, вам может захотеться пройти курс молодого бойца-ниндзя или даже сыграть на вылет, но вам не нужна глобальная аудитория, пока вы пытаетесь разобраться что к чему. Нет ничего позорного в том, чтобы пытаться сделать что-то и терпеть неудачи, но, работая над проектом, вы, скорее всего, не захотите, чтобы происходящее на стадии разработки было общедоступно. Находясь в разработке, проект может иметь проблемы с безопасностью, до которых руки еще не дошли, поэтому размещение его на рабочем сервере может поставить сервер под угрозу.

Разработка локально может сэкономить время и часто является отличным ускорителем производительности. Работая локально, вы не имеете необходимости подключаться к Интернету для тестирования кода. Ваш проект — вещь в себе на вашем компьютере. Это также означает, что вам не нужно перемещать файлы на удаленный сервер, чтобы проверить их. Вам достаточно сохранить внесенные правки и обновить браузер. Можно добавить время на ожидание соединения по FTP.

Если вы разрабатываете новую тему, то можете протестировать ее, используя разные наборы контента. Например, вы создаете индивидуальную тему для отдельного проекта с готовым контентом, но также хотите убедиться, что в будущем новый контент, добавленный на этот сайт, будет выглядеть надлежащим образом. Или же вы хотите выпустить тему для репозитория WordPress. Разрабатывая тему на локальном компьютере, вы можете использовать контент, отличный от контента рабочего сайта, чтобы убедиться, что все форматируется нужным образом. Это часть конфиденциальности локальной разработки. Данная концепция будет рассмотрена в этой главе подробнее.

Локально вы можете запускать несколько копий WordPress. Более того, каждая копия может быть разной версией WordPress. Это позволит отследить изменения в ядре WordPress и убедиться, что код будет продолжать работать в следующих редакциях. Например, вы можете тестировать тему или плагин на локальном сайте с текущей версией WordPress, но на вашем компьютере также может быть еще один сайт на WordPress, представляющий собой бета-версию или ежедневно отслеживающий изменения. Это поможет вам быть в курсе изменений ядра WordPress, которые могут повлиять на ваш проект.

Есть много преимуществ и причин для разработки локально. Для конкретного разработчика могут существовать и другие причины помимо изложенных нами преимуществ: конфиденциальности, безопасности и гибкости. Каждый разработчик должен самостоятельно проанализировать эти причины и решить, что для него более эффективно: риск или дополнительные шаги. В конце этой главы будут затронуты некоторые из основных проблем, возникающих при разработке локально и переносе вашего проекта на работающий сервер.

Установить локальную среду разработки для WordPress удивительно легко. Для этого нужны веб-сервер с интерпретатором PHP и база данных MySQL. Все эти инструменты находятся в открытом доступе.

Инструменты для администрирования компонентов

Подумайте о том, что необходимо для WordPress, и составьте список компонентов. WordPress — это веб-приложение. Значит, вам потребуется веб-сервер. WordPress работает на PHP, языке программирования для Интернета. Значит, веб-сервер должен поддерживать PHP. Хорошим (и очень популярным) универсальным веб-сервером является Apache, хотя есть множество других рабочих вариантов, в том числе Microsoft IIS и Nginx. Для WordPress версии 3.2 требуется версия PHP не ниже 5.2.4. В идеале вам нужен веб-сервер, который поддерживает преобразование URL, чтобы работали постоянные ссылки. У Apache для этой цели есть модуль `mod_rewrite`.

WordPress также требуется база данных для хранения контента сайта. В качестве базы данных WordPress поддерживает только MySQL, и для версии 3.2 версия MySQL должна быть 5.0 или выше. Кроме того, PHP должен быть снабжен надлежащими библиотеками MySQL для подключения к базе данных. Также вам понадобится клиент для управления базой данных.

Установка инструментов для разработки

Сначала список выглядит обескураживающе. Хотя многие из нас рассматривают WordPress как платформу, на которой строятся проекты, WordPress в свою очередь основан на платформе. Обычно она называется LAMP (Linux, Apache, MySQL и PHP). Она стала основой для множества интернет-проектов, в том числе для

Facebook. И именно она нужна для WordPress. То есть сообщество WordPress не единственное в своем роде с подобными требованиями.

Как уже было сказано, этот фундамент называется LAMP, где L соответствует Linux. Если Linux установлен в качестве операционной системы на вашем компьютере, вы можете установить LAMP, используя систему управления пакетами вашего дистрибутива Linux. Например, если вы работаете на Debian или на его производной, вам нужно запустить `apt-get install apache`, чтобы установить веб-сервер Apache. Обычно практикуется установка phpMyAdmin в качестве клиента MySQL, то есть нужно запустить `apt-get install phpmyadmin`. phpMyAdmin — это веб-приложение, требующее наличия Apache, PHP и MySQL. Поскольку это клиент MySQL, он также загрузит необходимые библиотеки, чтобы устанавливать связь между PHP и MySQL.

Более чем вероятно, что вы не используете Linux в качестве операционной системы. Вы можете установить все компоненты по отдельности, а затем соединить их, чтобы они работали. Это сложно. К счастью для нас, не перевелись еще энтузиасты, которые собрали несколько пакетов, делающих установку и настройку платформы LAMP простой. Пакеты существуют для различных операционных систем.

Если вы работаете с Mac OS X, то можете использовать установщик MAMP. Думаем, вы уже догадались, что аббревиатура обозначает: Macintosh, Apache, MySQL и PHP. Вы можете скачать MAMP по ссылке http://www.mamp.info.

Скачав MAMP, распакуйте его и установите как любое другое приложение для Mac. Переместив его в папку Программы (Applications), вы можете запустить MAMP и открыть контрольную панель. Она управляет всей платформой MAMP, в том числе вашими настройками. Единственное, что нам не нравится в MAMP, это то, что здесь не используется порт по умолчанию для Apache. Стандартным портом для веб-серверов является 80, и браузеры знают это, поэтому-то вы и не видите `80` в адресной строке. Однако у MAMP портом по умолчанию является 8888. Это означает, что, пытаясь попасть на локальный веб-сервер, вы должны будете указывать в браузере http://localhost:8888. Имейте это в виду, поскольку примеры в этой книге рассчитаны на стандартный порт 80.

Если вы работаете с Windows, у вас есть две возможности, а именно WAMP и XAMPP. WAMP предназначен специально для Windows и доступен по ссылке http://wampserver.com. Аббревиатура WAMP с очевидностью означает Windows, Apache, MySQL и PHP. XAMPP работает под Windows, но является межплатформенным. Его можно скачать по ссылке http://www.apachefriends.org. X в аббревиатуре XAMPP означает межплатформенность, а дополнительная P появилась потому, что XAMPP включает в себя PERL, еще один язык программирования. Оба предложенных варианта хороши.

Скачайте и установите WAMP подобно любому другому приложению для Windows. После установки в области уведомлений Windows (Windows system tray) появится новая иконка для WAMPSERVER, работающая как вызов панели управления.

Обратите внимание, что этот фундамент на самом деле является несколькими приложениями, слаженно работающими вместе, чтобы обеспечить вам платформу

для веб-разработки, способную поддерживать WordPress. Установщики WAMP и MAMP — просто инструмент автоматизации, чтобы связать эти компоненты для общей цели. Каждое отдельное приложение включает в себя отдельные файлы настройки, что позволяет подогнать его под ваши требования. Некоторые из наиболее распространенных настроек будут рассмотрены далее в этой главе.

Добавление WordPress в локальную установку

Теперь, имея рабочую платформу, вам нужно установить WordPress. Тут, пожалуй, вам захочется подумать о том, как вы собираетесь использовать локальную среду разработки. Нужна ли вам только одна копия WordPress? Если не одна, то собираетесь ли вы использовать подпапки или же будете делать отдельные сайты с использованием виртуальных хостов? Будете ли вы использовать функциональность WordPress Multisite для нескольких сайтов? В следующем разделе мы рассмотрим некоторые из этих возможностей. А пока пойдем простым маршрутом и сделаем один сайт на WordPress.

Чтобы установить WordPress, вы можете использовать тот же метод управления исходным кодом с использованием Git или Subversion, который показан в главе 2. Или же вы можете прибегнуть к традиционному варианту скачивания установочных файлов по ссылке http://wordpress.org.

Так или иначе, получив файлы ядра WordPress, вы должны поместить их в корневую папку документов веб-сервера. В случае с MAMP эта установка делается в контрольной панели MAMP Установки (Preferences) ▸ Apache. Вы можете принять вариант по умолчанию или установить ту корневую папку документов, которую хотите. Пользователи Mac обычно помещают ее в папку Сайты (Sites).

В WAMP корневая папка документов определяется как `c:\wamp\www`. Вы можете быстро перейти в эту папку, используя параметр `www directory` при вызове иконки WAMPSERVER.

Скопируйте файлы ядра WordPress в нужную корневую папку документов на вашем компьютере.

Теперь запустите веб-браузер и пройдите по ссылке http://localhost. Не забывайте, что если ваш локальный браузер не работает со стандартным портом, вам может понадобиться добавить его в адресную строку. Также, если вы скопировали WordPress в подпапку корневой папки документов, может понадобиться добавить суффикс к URL — например, http://localhost/ddamstra/Documents/www.

Если веб-сервер и сервер базы данных настроены правильно, WordPress создаст базу данных и отредактирует конфигурационные файлы, после чего вы увидите первую страницу установки WordPress, как это показано на рис. 3.1.

Как и всегда при установке WordPress, вам понадобится задать параметры доступа к базе данных. И WAMP, и MAMP снабжены phpMyAdmin для управления MySQL. Используйте панель управления WAMP или MAMP, чтобы попасть в phpMyAdmin и задать настройки.

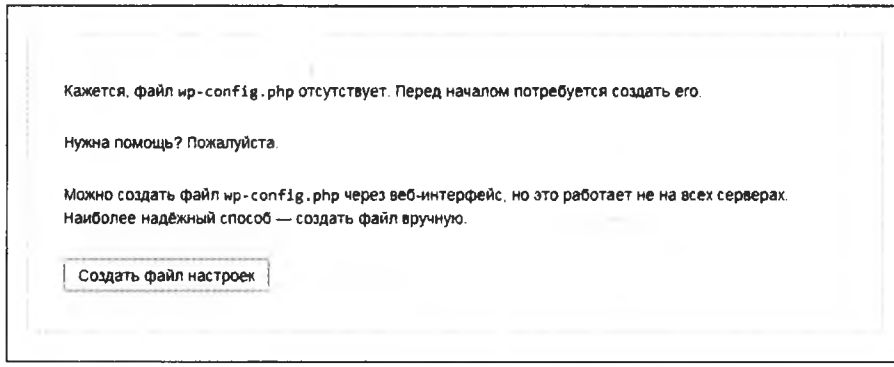


Рис. 3.1. Установка WordPress

Наконец, выполните знаменитую 5-минутную установку WordPress, как это описано в главе 1.

Если у вас возникли проблемы с тем, чтобы заставить локальную среду разработки функционировать, поищите помощи в соответствующих сообществах и в документации. Несмотря на то что пакеты подготовлены так, чтобы без проблем установить различные компоненты, двух одинаковых компьютеров не бывает, а управление настройкой этих различных составляющих лежит за пределами темы данной книги, поскольку затрагивает WordPress лишь по касательной.

Детали настройки

В предыдущем разделе мы посмотрели, как установить локальную среду разработки. Хотя обзор был поверхностным, основные моменты стали понятны. В этом разделе мы поговорим о расширении среды и рассмотрим некоторые тонкости, которые помогут получить максимум от работы локально. Некоторые указания вновь будут касаться платформы LAMP.

Здесь мы изучим параметры настройки более детально. Этот раздел расскажет вам об управлении древом файлов, которое видит веб-сервер, об активации отладки и о создании имен виртуальных серверов.

Управление древом документов веб-сервера

В предыдущем разделе мы приняли корневую папку для документов для Apache по умолчанию. Однако по различным причинам это может быть не самое лучшее место для рабочего процесса или систем резервного копирования.

Например, в своей экспериментальной лаборатории с многочисленными веб-разработчиками вы можете переназначить корневую папку документов Apache на `c:\www`. В таком случае корневые папки документов у каждого разработчика идентичны, и это будет папка верхнего уровня, легкодоступная. И наоборот: на своем

личном ноутбуке вы можете переназначить корневую папку документов на `C:\Users\ddamstra\Documents\www`, поскольку при подключении к домашней локальной сети происходит резервное копирование папки Documents.

Будьте осторожны при внесении изменений. Как мы уже неоднократно говорили, речь идет о работе множества составных частей, и достаточно вывести из строя одну из них, чтобы столкнуться с серьезными последствиями. MAMP позволяет изменять корневую папку документов через панель управления. С помощью WAMP вы редактируете конфигурационный файл для Apache. Этот файл называется `httpd.conf`, и его можно найти в панели управления WAMPSERVER в пункте всплывающего меню Apache.

Измените строку считывания корневой папки документов (DocumentRoot), чтобы указать выбранное местоположение, как это показано на рис. 3.2.

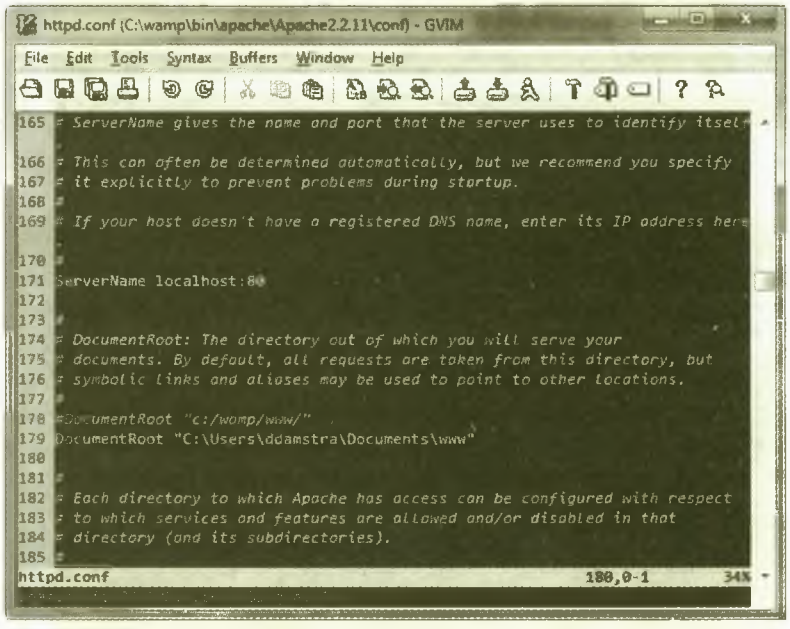


Рис. 3.2. Корневая папка документов Apache

Вам также понадобится сменить путь к директории, чтобы получить соответствие, как это показано на рис. 3.3.

При использовании панели управления WAMP вам потребуется перезапустить Apache (или все службы), чтобы внесенные изменения вступили в силу. Если раньше файлы находились в старой корневой папке документов, их нужно переместить в новую, чтобы сделать доступными.

Улучите момент, чтобы внимательно посмотреть, что вы публикуете в корневой папке документов. Едва ли вам хочется публиковать личные или конфиденциальные данные. Подумайте, какую систему контроля исходного кода вы собираетесь

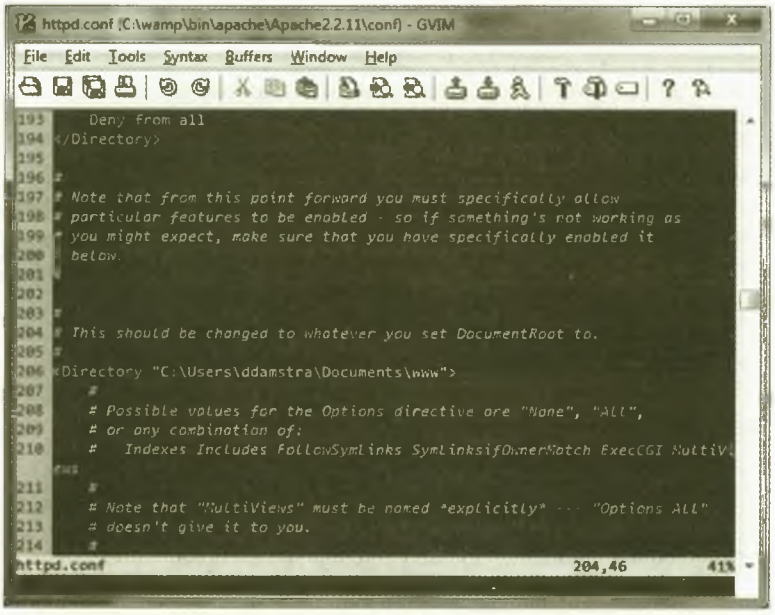


Рис. 3.3. Путь к директории Apache

Рис. 3.4. Блокировка файлов .svn для Apache

использовать. Является ли эта система также частью вашей стратегии разработки? Если вы используете публичный репозиторий, такой как GitHub, убедитесь, что вы не поместили туда файл `wp-config.php` и не выставили на всеобщее обозрение свои пароли. Точно так же, если ваша среда разработки доступна в локальной сети,

посмотрите, где вы сохраняете конфигурационные файлы с важной информацией. Некоторые системы контроля исходного кода, в особенности Subversion, хранят редакции в виде открытого текста в файлах в папке вашего проекта, потенциально демонстрируя данные для авторизации. Такое случалось с нами неоднократно при выполнении тестовых заданий по внутреннему проникновению, и все это — часть стандартной настройки Apache. Вы можете настроить Apache так, чтобы он не обслуживал директории `.svn`, добавив строки, показанные на рис. 3.4, в файл `httpd.conf`.

Информации для отладки

При разработке локально вы хотите избавиться от максимального количества потенциальных ошибок и предупреждений. Как минимум вы должны знать о них. Необходимо выставить условия для ошибок в PHP наиболее жестко, что увидеть эти ошибки и исправить их.

Как уже обсуждалось в главе 1, это прямая противоположность тому, что вы хотели бы видеть на рабочем сервере. Там вам нужно скрыть все ошибки от посетителей. На вашем компьютере единственный посетитель — это вы, и вы хотите увидеть все ошибки, потому что это то, над чем вы работаете.

Уровень ошибок PHP выставляется в файле `php.ini`. Используя WAMP, вы можете обратиться к этому файлу через панель управления WAMP, пункт всплывающего меню PHP. Задайте директиве сообщения об ошибках значения `E_ALL` и `E_STRICT`, как это показано на рис. 3.5.

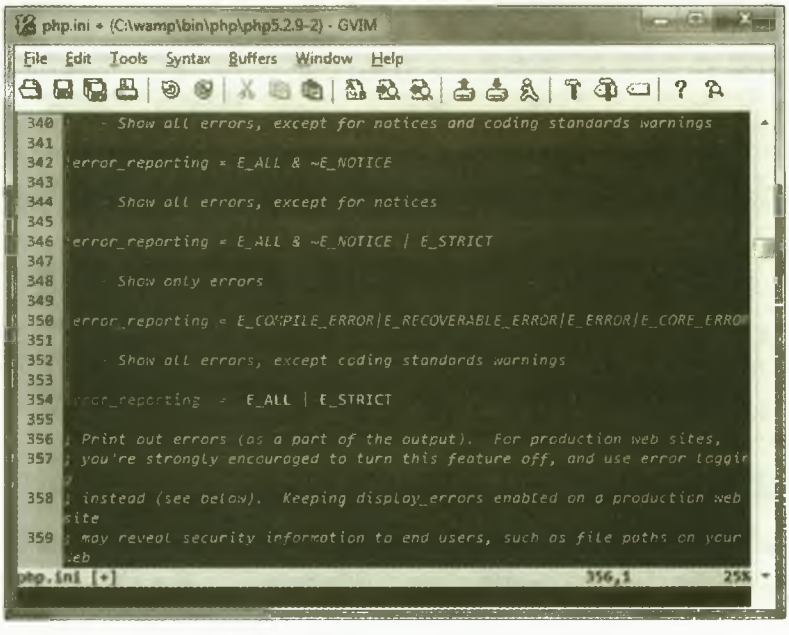


Рис. 3.5. Уровень ошибок PHP

Вплоть до PHP 5.4 строгие предупреждения и замечания не были включены в уровень E_ALL. Приписывая директиве сообщения об ошибках указанные выше параметры, вы гарантированно увидите столько сообщений об ошибках, сколько возможно, а написание кода с целью уменьшения количества этих замечаний позволит вам обеспечить максимальную функциональную совместимость PHP. Повторяем: вам необходимо перезапустить Apache, чтобы изменения вступили в силу.

Как уже упоминалось ранее, при разработке на одной операционной системе и последующем внедрении в другую нужно помнить, что не у всех систем один и тот же PHP API. Например, значения параметра PHP $_SERVER[] на машинах с Windows не совпадают с его значениями на машинах с Linux. Windows нечувствительна к регистрам в файловой системе, чего не скажешь о Linux. Разработчики должны помнить, что конечная система может не быть их системой для разработки. Вот почему нужно, чтобы отклонения могли быть устранены до внедрения.

При разработке локально активируйте отладку WordPress. Подобно сообщениям об ошибках PHP, это позволяет разработчику видеть изъяны WordPress и работать с ними. Поэтому этот параметр всегда должен быть отключен на рабочем веб-сайте.

Активируйте отладку WordPress, отредактировав файл wp-config.php и установив значение параметра WP_DEBUG на true, как это показано на рис. 3.6. В отличие от предыдущих настроек Apache и PHP, которые касались всех сайтов на вашем компьютере, эта применяется только к копии WordPress.

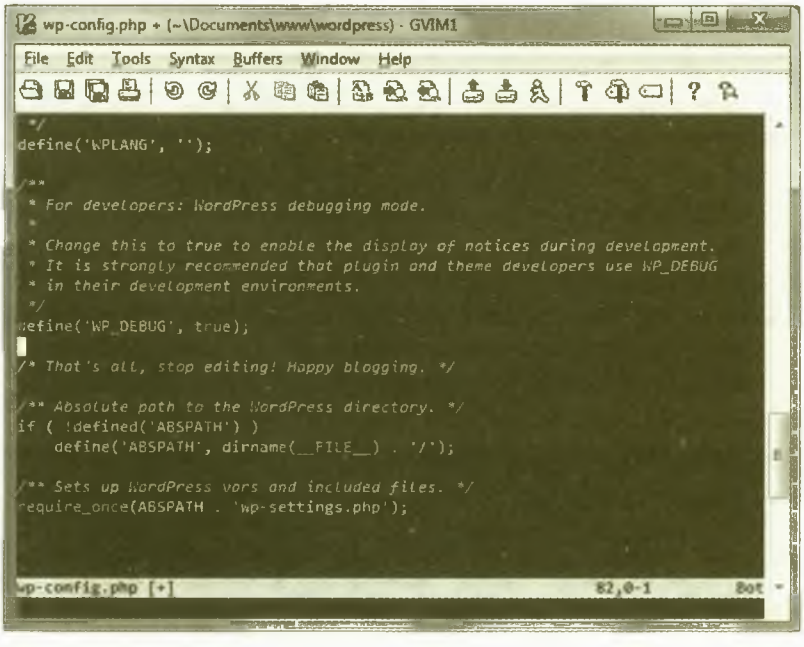


Рис. 3.6. Отладка WordPress

Работа с локальной и рабочей базой данных

Сразу после установки WordPress настроен на одну базу данных. Работая локально, вы хотите, чтобы находящийся в разработке сайт подключался к локальной копии MySQL, чтобы не испортить рабочую базу данных. Другими словами, эта одна из причин работы локально.

Самый распространенный метод — присвоить хосту базы данных значение `localhost` и задать данные авторизации для MySQL и имя таблицы локально такими же, как на рабочем сайте. Но это плохо с точки зрения безопасности.

Марк Джекит предлагает альтернативное решение, позволяющее и рабочему и локальному компьютеру иметь свой набор данных для авторизации в базе данных: http://markjaquith.wordpress.com/2011/06/24/wordpress-local-dev-tips/. По сути, разработчик изменяет файл `wp-config.php` таким образом, чтобы данные для авторизации, существующие только на машине разработчика, имели приоритет. После этого полученный файл `wp-config-local.php` игнорируется в системе контроля исходного кода, что позволяет каждому разработчику иметь свои локальные данные для авторизации, причем этот файл никогда не попадет в рабочую среду.

Создание имен виртуальных локальных серверов

Изначально вы устанавливаете WordPress в корневую папку документов вашей локальной копии Apache. Если вам нужно более одного локального веб-сайта, вы можете установить каждый сайт в свою папку. Это работает и используется для многих сайтов в разработке. Однако вы также можете настроить каждый веб-сервер таким образом, чтобы он отвечал на «фальшивое» локальное имя домена. Иногда при перемещении в рабочую среду этот метод облегчает преобразование из версии в разработке в рабочую.

Вот как это работает с применением некой сетевой магии. Каждый знает полные имена наиболее распространенных доменов верхнего уровня, таких как `.com`, `.net`, `.org` и многих других. Эти имена доменов работают через систему DNS, в которой браузер запрашивает у этих доступных через Интернет DNS-серверов IP-адрес веб-сайта домена, который вы ввели.

Однако ваш браузер использует преобразователь DNS, чтобы проверить сначала локальный файл и увидеть, есть ли в нем предустановленные правила присвоения. Файл называется *hosts file*. Вы можете использовать этот файл и подходящую конфигурацию Apache, чтобы заставить свой компьютер подключаться к локальным сайтам с фальшивыми полными доменными именами.

Есть разные способы решить этот вопрос. Некоторые разработчики устанавливают доменное имя сайта, над которым они работают, в качестве локальной рабочей станции, прерывая запросы DNS. Это означает, что пока они не уберут эти изменения,

у них не будет доступа к работающему сайту, все запросы будут идти на локальный сайт. Например, вместо того чтобы получать публично доступный IP-адрес для `mirmillo.com`, соответствующие запросы перехватывают и переадресуются IP-адресу `localhost`, которые всегда 127.0.0.1.

Другой вариант — снабдить разрабатываемый сайт фальшивым именем, которое легко заменить в SQL во время фазы внедрения. В таком случае вы обозначаете локально разрабатываемый сайт как `mirmillo.local`, что является неверным доменом верхнего уровня (на данный момент). Таким образом, мы можем выходить на `mirmillo.com`, используя традиционный DNS, и по-прежнему работать с нашей локальной разрабатываемой версией, выходя на нее как на `mirmillo.local` через браузер. Этому примеру мы и будем следовать в книге.

Сначала вам необходимо настроить Apache на поддержку виртуальных хостов. Настройка зависит от вашей копии Apache. При использовании WAMP первым шагом было установление виртуального хоста в Apache. Это делается посредством редактирования файла `httpd-vhosts.conf`, находящегося по адресу `C:\wamp\bin\apache\Apache2.2.11\conf\extra`. Пример по умолчанию снабжен двумя образцами виртуальных хостов. Превратите один из существующих образцов в виртуальный хост `localhost`. Затем отредактируйте второй образец так, чтобы он соответствовал настройкам, необходимым для вашей локальной установки, такой как `mirmillo.local`, как это показано на рис. 3.7.

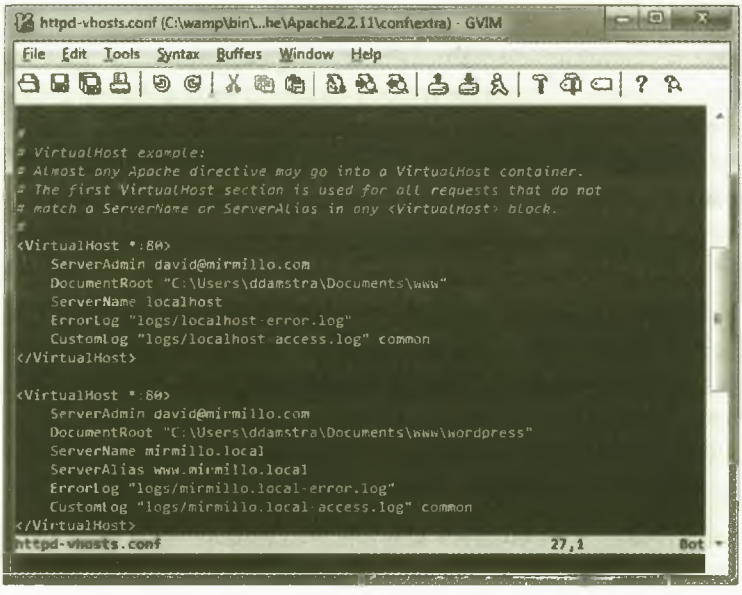


Рис. 3.7. Виртуальный хост mirmillo.local

Теперь необходимо указать Apache использовать этот файл. Отредактируйте файл `httpd.conf`, как вы уже делали ранее. Как показано на рис. 3.8, раскомментируйте строку, чтобы использовать настройки виртуального хоста.

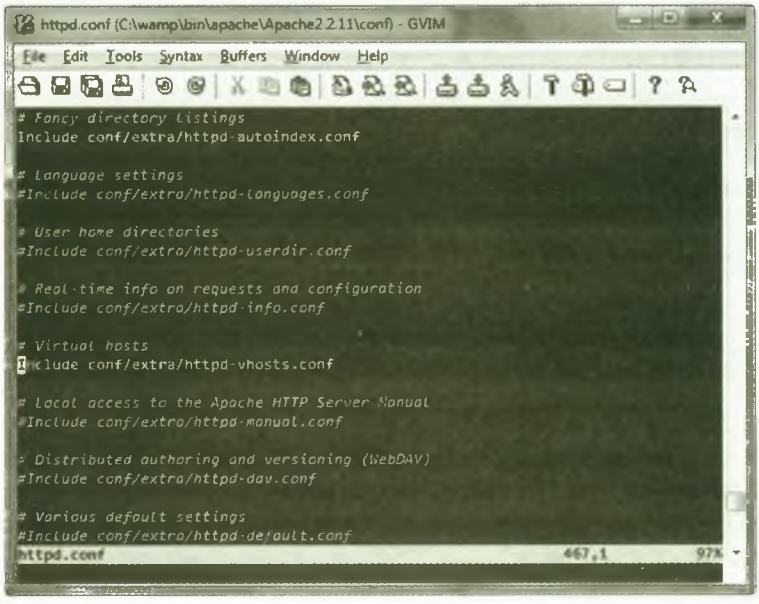


Рис. 3.8. Apache использует настройки виртуального хоста

После этого отредактируйте файл hosts. В Mac OS X он находится здесь: `/private/etc/hosts`, в Linux — здесь: `/etc/hosts`, в Windows — тут: `C:\Windows\System32\drivers\etc`. Этот файл состоит из пар IP-адресов и имен доменов. Как показано на рис. 3.9, вы можете присвоить новый адрес `mirmillo.local`.

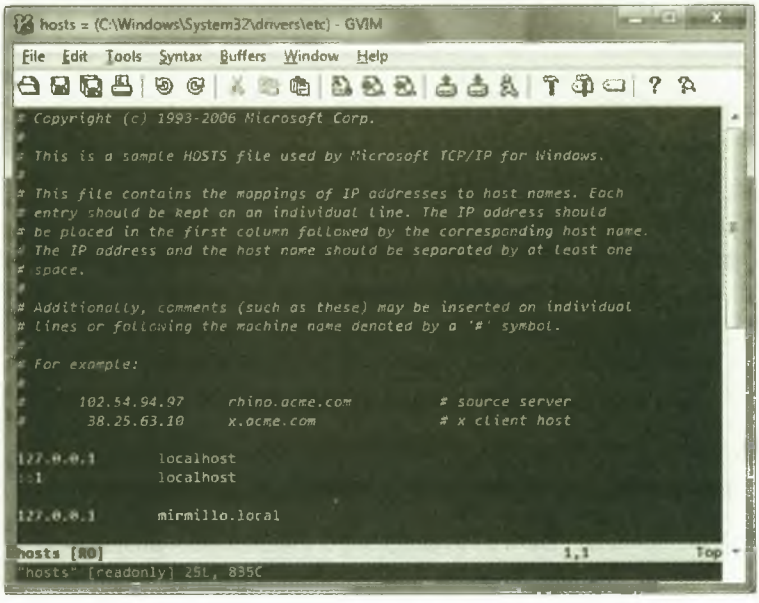


Рис. 3.9. Присвоение адреса виртуальному хосту в файле Hosts

Теперь перезапустите Apache и перейдите на http://mirmillo.local, чтобы окончить установку WordPress, как уже было описано в главе 1.

Разработка тем и плагинов локально

Если вы разрабатываете тему, одним из преимуществ разработки локально является возможность не использовать контент работающего сайта. В самом деле, если вы разрабатываете тему, которую планируете выпустить в свет, вам необходимо использовать «фиктивный» контент, чтобы убедиться, что дизайн подходит для разнообразного содержимого. Например, вы можете использовать шаблонный контент WordPress, доступный по ссылке http://codex.wordpress.org/Theme_Unit_Test. Есть несколько альтернативных файлов, импортирующих образцы содержимого. Один из них предоставляется WPCandy по ссылке http://wpcandy.com/made/the-sample-post-collection. Но эксперты тем WordPress будут использовать собственные варианты, чтобы убедиться, что ваша тема может быть помещена в репозиторий. Вы можете найти перечень требований к теме для репозитория по ссылке http://codex.wordpress.org/Theme_Development_Checklist. Этот вопрос также рассматривается в главе 9 «Разработка тем».

Скажем, вы разрабатываете тему и хотите проверить ее с помощью шаблонного содержимого, упомянутого в предыдущем параграфе, но вам также нужно охватить специфический контент сайта, для которого вы эту тему разрабатываете. Здесь хорошо использовать WordPress Multisite. WordPress Multisite подробно, включая установку, рассматривается в главе 10. Но если вы установили его локально, WordPress Multisite позволит вам использовать одни и те же темы и плагины для множества сайтов на WordPress в сети WordPress. Мы настроили все таким образом, что один из сайтов на WordPress имеет шаблонное содержимое. Затем мы создали второй сайт со специфическим контентом. Далее нужно сделать разрабатываемую тему доступной по Сети и активировать ее на обоих сайтах. Это позволит вам видеть в браузере два разных набора контента для WordPress, редактируя только один набор файлов темы.

Точно так же, разрабатывая новый плагин, проверьте его в WordPress Multisite, чтобы убедиться, что он работает. Вы также можете установить на вашей машине несколько виртуальных хостов с разными версиями WordPress, включая несколько предыдущих редакций и ту, что находится в разработке, чтобы убедиться, что ваш плагин будет работать и после следующего обновления системы. Хотя мы рекомендуем пользователям всегда иметь актуальную версию WordPress, реальность такова, что некоторые сайты продолжают работать на предыдущих версиях: из-за ограничений хостинга, незнания или лени администраторов. Важно убедиться, что ваш плагин будет работать, если люди используют его. Также обратитесь к главе 8 «Разработка плагинов».

Теперь, после того как ваш новый проект заработал локально и вы устранили все ошибки и замечания WordPress и PHP, самое время переместить его на рабочий сервер. В следующем разделе вы изучите некоторые проблемы и их решения при переносе кода в рабочее состояние.

Внедрение локальных изменений

Сначала разделим различные типы объектов, которые мы внедряем. Есть код, который может быть кодом плагина, темы или ресурсов темы. Есть контент, то есть то, что обсуждается, будучи опубликованным в виде записей или страниц сайта; контент хранится в базе данных. Наконец, есть настройки, которые также хранятся в базе данных.

Внедрить код легко. Разработчики делают это каждый день. Одним из преимуществ PHP и WordPress является то, что вы, в общем-то, можете просто скопировать код в корневую папку документов, и он запустится при следующем запросе. Внедрить код просто, и вы можете использовать для этого FTP-клиент. Но, пожалуйста, используйте по возможности SFTP, поскольку это безопасный протокол в отличие от FTP.

С внедрением контента и настроек дело обстоит сложнее. WordPress использует полные ссылки внутри всего контента. Поэтому каждая ссылка (атрибут ссылки HREF) и элементы меню содержат полное доменное имя. Точно так же от полного доменного имени, на котором установлен WordPress, зависят и все настройки сайта. Вы не можете просто взять дамп базы данных и переместить его.

Однако есть промежуточный шаг, который состоит в смене доменных имен в экспортируемой базе данных, перед тем как она будет импортирована на рабочий сайт. Будьте осторожны, чтобы не перезаписать свежий контент на рабочем сайте контентом с разработанной версии. Как именно вы будете делать это, зависит от ваших конкретных нужд, но в целом этот процесс схож с перемещением сайта с одного домена на другой. Процедура исчерпывающе задокументирована на различных веб-сайтах, в кодексе WordPress http://codex.wordpress.org/Moving_WordPress и во многих руководствах. Это только один из методов.

Вот краткое описание того, как работает процесс, при условии, что вы хотите переместить весь контент из базы данных для разработки на работающий сайт.

Задача состоит в том, чтобы удалить все полные ссылки на сайт в разработке из контента. После того как перемещен контент из версии разработки, весь следующий контент, который добавится на рабочий сайт, будет снабжаться полными ссылками, но этот метод состоит в том, чтобы сделать все URL относительными, после чего они станут работать и на сайте в разработке, и на рабочем сайте.

Для этого используйте плагин wp-DBManager, написанный Лестером Ченом. Данный плагин позволяет делать резервные копии баз данных, а также выполняет SQL-запросы. Вы также можете использовать встроенный экспорт баз данных WordPress и phpMyAdmin.

Представим себе, что перемещение осуществляется с локального сайта в разработке `mirmillo.local` на рабочий сайт `mirmillo.com`. Здесь пригодится описанная ранее возможность использования «фальшивого» доменного имени виртуального хоста.

Используя плагин, делаем резервную копию работающего сайта. Скачиваем и сохраняем ее на случай, если что-то пойдет не так.

Затем со страницы SQL плагина посылаем запросы, показанные на рис. 3.10, чтобы обновить URL в контенте. По сути, вы удаляете имя домена из URL в коде HTML.

Теперь экспортируем содержимое с сайта в разработке. Функция экспорта содержимого находится в Консоли WordPress Инструменты ▸ Экспорт. Скачиваем файл. Это и есть наш перемещаемый контент с уже относительными путями.

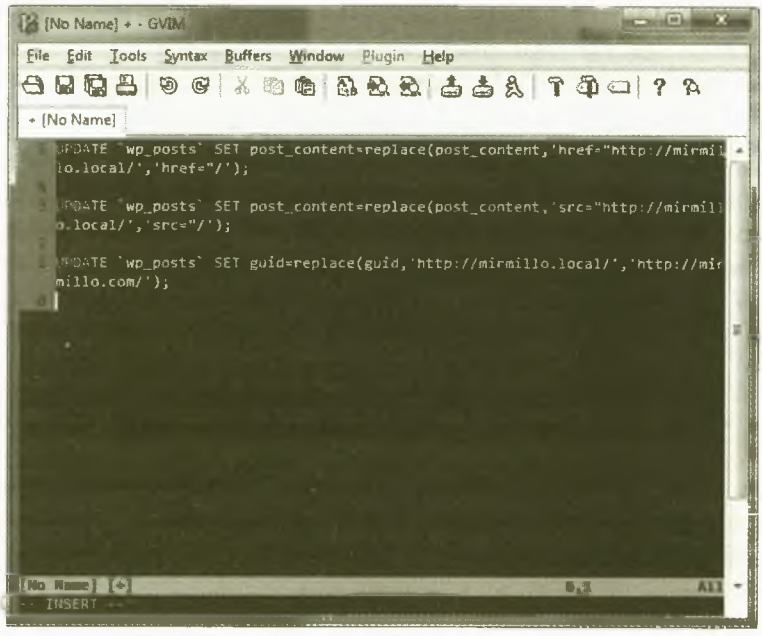


Рис. 3.10. Запросы SQL для удаления доменных имен

Импортируем это содержимое на наш рабочий сайт. Функция импорта содержимого находится в Консоли WordPress Инструменты ▸ Импорт. Еще раз: будьте осторожны, не перезапишите новый контент или содержимое, которое вам хотелось бы сохранить.

На самом деле процесс не сложный, но требует некоторого планирования и координации. Есть разработчики, которые корпят над инструментарием, способным облегчить эту задачу. В частности, мы пристально наблюдаем за RAMP от Crowd Favorite Алекса Кинга, доступным онлайн http://crowdfavorite.com/wordpress/ramp/. Хотя мы еще не испытывали его, выглядит он многообещающе. Сложность в том, что всегда, когда WordPress используется как система управления контентом, пользователи могут и будут заходить на рабочий сайт и вносить изменения. В этом все дело. Если это происходит, контент версии в разработке устаревает. В конечном счете цель — получить способ синхронизации баз данных WordPress: рабочей,

промежуточной и разрабатываемой и иметь возможность разрешить конфликт. Здесь нет верного решения, но, похоже, это тот вопрос, над которым ломают голову многие разработчики.

Резюме

Эта глава охватила некоторые мотивы и процессы, необходимые для нормального хода разработки. Кроме того, в ней было рассказано, как превратить локальную среду разработки WordPress в вашу личную песочницу. Наконец, был рассмотрен процесс перемещения сайта в разработке на рабочий сервер. Следующая глава будет посвящена файлам ядра WordPress и обзору работы этой системы.

4

Обзор ядра

В этой главе:

- Изучение файлов ядра WordPress
- Использование файлов ядра как справочника
- Работа с Кодексом WordPress
- Понимание встроенной документации

Чтобы понять, как правильно расширять WordPress, нужно сначала изучить работу его ядра. Это поможет вам узнать, какой инструментарий, доступный в ядре WordPress, может облегчить вашу жизнь. WordPress избавляет разработчиков от скучного кодирования и помогает решить большинство логических проблем.

Ядро WordPress — лучший источник для изучения работы WordPress. Красота программного обеспечения с открытым кодом в том, что весь код находится в вашем распоряжении. Если вы не уверены в том, как функционируют те или иные аспекты WordPress, просто начните копаться в коде! Там есть все ответы, надо только их найти и понять.

Что есть в ядре?

Ядро WordPress — это набор файлов, являющихся частью оригинального скачанного программного обеспечения WordPress. Они являются обязательными файлами, без которых WordPress не сможет функционировать надлежащим образом. Подразумевается, что файлы ядра изменяются только в том случае, когда вы обновляете WordPress до новой версии.

Ядро не включает в себя индивидуальные файлы для плагинов, тем, настройки баз данных, файл `.htaccess` и т. д. В ядре также не содержатся медиафайлы, которые

вы загружаете в WordPress. Практически любые файлы, добавленные в WordPress после установки, считаются находящимися вне ядра.

Файлы ядра WordPress являются в первую очередь файлами PHP, но среди них также есть файлы CSS, JavaScript, XML, HTML и файлы изображений. Они контролируют все в WordPress, включая то, как генерируется контент страниц для отображения, загрузку настроенной темы и плагинов, загрузку всех параметров и настроек и много чего еще. Вкратце, ядро содержит несколько основных типов функций:

- **Записи, страницы и индивидуальный контент**. Создание, хранение, выборка и взаимодействие с большей частью пользовательского контента WordPress. Обсуждение цикла, контролирующего отображение и упорядочивание содержимого, в главе 5 основано на этих функциях.
- **Метаданные**. Все от меток и категорий до создаваемых пользователями таксономий. Используемые модели данных изучаются в главе 7.
- **Темы**. Поддержка функций для тем WordPress. Разработка темы и ее соотношение с этими функциями обсуждается в главе 9.
- **Действия, фильтры и плагины** — инфраструктура для расширения WordPress, подробнее рассматривается в главе 8.
- **Пользователи и авторы**. Создание и управление контролем доступа к сайту, а также вопросы безопасности на уровне предприятия — в главах 12 и 14.
- **Фиды, форматирование и комментарии**. Все это обсуждается в книге по мере необходимости.

Мы коснемся всех этих вопросов по мере изучения ядра WordPress. Относитесь к этой главе как к путеводителю, рассказывающему о том, «как» изучать ядро WordPress. Это полевой справочник по документации Кодекса WordPress для пользовательских дискуссий и объяснений. Она также необходима для удобного поиска по ядру в дополнение к представленному здесь обзору функций. Сюда не включен исчерпывающий список всех функций WordPress: и потому что он меняется и расширяется по мере непрестанной разработки ядра WordPress, и потому что наша цель здесь — показать оценку разработчика и внедрителя, а не суммировать Кодекс.

WordPress поступает в комплекте с двумя плагинами: Akismet и Hello Dolly. Они находятся в директории плагинов внутри `wp-content`. Несмотря на то что оба этих плагина являются частью скачиваемого пакета файлов ядра WordPress, они не считаются функциями ядра, поскольку должны быть активированы для начала работы и легко могут быть удалены.

WordPress также идет в комплекте с тремя темами: Twenty Ten, Twenty Eleven и Twenty Twelve. Twenty Twelve является темой по умолчанию на свежеустановленной копии WordPress. Так же как и в случае с плагинами, файлы этих тем не считаются функциями ядра, поскольку их легко можно заменить любой темой, которую вы хотели бы использовать на вашем сайте.

Использование ядра как справочника

Чтобы использовать ядро WordPress как справочник, вам нужно понимать, чего ждать от файлов ядра. Большинство файлов ядра WordPress содержат документацию в виде комментариев к коду. Обычно комментарий к коду отображается в заголовке файла и дает общее представление о том, что за файл вы просматриваете.

Чтобы увидеть это своими глазами, откройте файл `wp-login.php`, расположенный в корневой директории WordPress. Вы заметите, что в верхней части файла содержится заглавный комментарий, описывающий функцию файла:

```
/**
 * WordPress User Page
 *
 * Авторизация, регистрация, переустановка пароля и другие настройки.
 *
 *
 * @package WordPress
 */
```

Все файлы ядра, кроме изображений, могут быть просмотрены с использованием текстового редактора. В зависимости от настроек программы по умолчанию вам может понадобиться сначала открыть текстовый редактор, а потом файл или же открыть файл напрямую. Также полезно использовать редактор, который выделяет синтаксис, то есть синтаксис PHP будет выделяться, что сделает чтение кода более легким.

Полный список совместимых текстовых редакторов есть в Кодексе WordPress.org http://codex.wordpress.org/Glossary#Text_editor.

Встроенная документация

Практически все файлы ядра WordPress содержат встроенную документацию в форме PHPDoc. PHPDoc — это стандартизированный метод описания использования функции в форме комментария PHP. Это значит, что каждая функция объясняется детально в следующем непосредственно за ней блоке комментария. Вот шаблон для документирования функции WordPress:

```
/**
 * Короткое описание
 *
 * Длинное описание
 *
 * @package WordPress
 * @since version
 *
 * @param    type    $varname    Описание
 * @return   type                Описание
 */
```

Это исключительно полезно для понимания работы функций. Комментарий включает в себя краткое и полное описание. В нем также содержится версия WordPress, в которой функция была добавлена. Это помогает определить новые функции, добавленные в WordPress после выпуска новой версии.

Доступные параметры также перечисляются вместе с типами данных параметров. Тип данных — это тип данных, которые необходимы параметру. Например, параметр ID будет, скорее всего, использовать тип данных `int` (целое). Оставшаяся часть информации — это возвращаемое значение. Тип данных возвращаемого значения также указывается.

Все новые функции, добавляемые в WordPress, также документируются с использованием этого шаблона. За дополнительной информацией по встроенной документации WordPress обратитесь к статье Кодекса: http://codex.wordpress.org/Inline_Documentation.

Поиск функции

Поиск функций в ядре — самый быстрый способ выяснить, как работает та или иная функция WordPress. Вы можете точно увидеть, какие параметры можно приписывать этой функции, а также что именно она делает и какие значения возвращает.

Для начала убедитесь, что вы скачали на компьютер самую последнюю версию WordPress. Вы будете искать по этим файлам как по справке WordPress. Откройте любой текстовый редактор, в котором есть поиск по файлам (рекомендуется TextPad для Windows и Textmate для Mac). Во время поиска функции удобнее исключить вызовы функции из результатов. Сделайте это, включив слово «функция» в начало поискового запроса, например `function wp_head`. Не все в WordPress является функциями, но с них неплохо начинать. Если вы не нашли ни одного соответствия, удалите слово `function` из запроса. Также не забудьте настроить редактор на поиск по всем файлам (`*.*`), а не только по файлам `.txt`.

Рассмотрим функцию `is_super_admin()`. Она используется для проверки того, является ли пользователь суперадминистратором в WordPress Multisite. Вам нужно точно знать, какие значения требует функция, чтобы ее использовать. Откройте текстовый редактор и поищите по всем файлам WordPress функцию `function is_super_admin`. Поиск должен дать один результат в `wp-includes/capabilities.php`:

```
function is_super_admin( $user_id = false ) {
```

Вы сразу же видите один параметр, который можно посылать этой функции: `$user_id`. Обратите внимание на встроенную документацию, расположенную непосредственно над этой функцией. В случае `is_super_admin()` она выглядит следующим образом:

```
/**
 * Определяет, является ли пользователь администратором сайта.
 *
 * @since 3.0.0
```

```
 *
 * @param int $user_id (Optional) ID пользователя. По умолчанию - текущий
пользователь.
 * @return bool True, если пользователь является администратором.
 */
```

Это исключительно ценная информация. Комментарий содержит краткое описание того, что делает функция. В данном случае: «Определяет, является ли пользователь администратором сайта». Здесь также отмечено, когда эта функция была добавлена (в версии 3.0.0). Имеется и информация о единственном параметре, в том числе его тип, за что он отвечает и то, что в данном случае он является необязательным. Помимо этого комментарий сообщает, каким будет ожидаемое возвращаемое значение. В данном случае функция вернет `True`, если пользователь является администратором сайта, и `False`, если нет.

Уже этого достаточно, чтобы понять, как работает функция, но давайте заглянем в код для лучшего понимания. Первые несколько строк выглядят следующим образом:

```
if ( $user_id )
    $user = new WP_User( $user_id );
else
    $user = wp_get_current_user();
```

Исходя из комментария PHPDoc, расположенного над функцией, вы знаете, что параметр `$user_id` не является обязательным, поэтому код показывает, что происходит, если параметр `$user_id` не задан для функции. Предыдущее предложение с оператором `if` проверяет, есть ли значение у переменной `$user_id`. Если оно присвоено, возвращаются пользовательские данные соответствующего класса пользователя WordPress, `WP_User` для введенного ID. Если переменная `$user_id` пустая, то вызывается функция `wp_get_current_user()`, чтобы получить пользовательские данные для пользователя, авторизованного в данный момент.

Затем, перед тем как начать обработку, функция проверяет, что данные `$user` действительно существуют, если их нет, возвращается значение `false`.

```
if ( ! $user->exists() )
    return false;
```

Теперь, после того как вы узнали, что данные пользователя `$user` существуют, необходимо проверить, действительно ли он является суперадминистратором:

```
if ( is_multisite() ) {
    $super_admins = get_super_admins();
    if ( is_array( $super_admins ) && in_array( $user->user_login,
        $super_admins ) )
        return true;
} else {
    if ( $user->has_cap('delete_users') )
        return true;
}
```

Давайте разделим предложение с `if` на части:

```
if ( is_multisite() ) {
```

Оператор `if` проверит, что Multisite запущен на WordPress, вызывая функцию `is_multisite()`. Суперадминистратор будет существовать только при наличии активированной функции Multisite на WordPress.

Когда WordPress определил, что Multisite работает, функция вызывает `get_super_admins()`, чтобы вернуть массив с данными всех суперадминистраторов в WordPress, используя код:

```
$super_admins = get_super_admins();
```

Теперь переменная `$super_admins` представляет собой массив имен пользователей всех суперадминистраторов. Следующая строка — самая важная в этой функции. Это код, который проверяет, что пользователь является суперадминистратором WordPress:

```
$super_admins = get_super_admins();
if ( is_array( $super_admins ) && in_array( $user->user_login, $super_admins ) )
    return true;
```

Перед тем как работать с массивом, неплохо убедиться, что переменная действительно в нем существует, используя функцию PHP `is_array()`. Вторая часть этой строки кода использует функцию PHP `in_array()`, чтобы проверить, что учетная запись пользователя существует в массиве суперадминистраторов. Если она есть, пользователь является суперадминистратором, и функция возвращает значение `true`.

Если упомянутая ранее проверка `is_multisite()` возвращает значение `false`, функция выполняет следующий код `else`:

```
} else {
    if ( $user->has_cap('delete_users') )
        return true;
}
```

Приведенный выше код проверяет, имеет ли пользователь возможность `delete_users` (удалять пользователей). По умолчанию эта возможность приписывается обычным учетным записям администраторов WordPress. Если Multisite деактивирован в WordPress, но вы являетесь администратором, то код вернет значение `true` при вызове функции `is_super_admin()`.

Последняя строка кода функции:

```
return false;
```

Этот код, по сути, говорит, что если любая из проверок функции `is_super_admin()` не удалась, возвращается значение `false`. Это скорее мера безопасности, гарантирующая, что значение `true` или `false` всегда будет возвращено.

После того как мы разобрали этот пример, должно стать понятнее, насколько полезным может быть код ядра WordPress. Вы в точности выяснили, как работает эта функция, изучив исходный код. Все ответы на ваши вопросы существуют внутри ядра, так что принципиально важно иметь хорошее понимание того, как использовать ядро для собственных нужд.

Исследуем ядро

В ядре WordPress есть несколько файлов, которые содержат многие из самых популярных функций WordPress. Эти функции используются для всех API WordPress и подходят для любого индивидуального плагина или темы. Следующие разделы подробно расскажут о файлах ядра, которые содержат ключевые части кода для работы с WordPress. Все файлы, перечисленные в следующем разделе, расположены в директории `/wp-includes`.

Functions.php

Файл `functions.php` содержит основные функции API WordPress. Они используются для простого взаимодействия с WordPress стандартизированными методами. Плагины, темы и ядро WordPress используют эти функции:

- `current_time` — возвращает текущее время на основе определенного типа.
- `force_ssl_login` — требует SSL (https) авторизации на WordPress.
- `wp_nonce_field` — отображает скрытое поле специальных значений для форм. Поле специальных значений используется в целях проверки при внесении и обработке данных в WordPress. Это критически важный шаг для безопасности кода.
- `absint` — конвертирует значение в неотрицательное целое.

Option.php

Файл `option.php` содержит основные параметры функций API WordPress. Функции используются в следующих целях:

- `add_option`, `update_option`, `get_option` — функции создания, обновления и отображения сохраненного параметра.
- `set_transient`, `get_transient`, `delete_transient` — функции создания, возврата и удаления временных параметров в WordPress. *Временный параметр* — это параметр с ограниченным сроком действия. По достижении соответствующего времени автоматически удаляется из WordPress.
- `add_site_option`, `update_site_option`, `get_site_option` — функции создания, обновления и отображения параметров сайта. Если активирован Multisite, функция возвращает сетевой параметр, если нет — параметр стандартного сайта.

Formatting.php

Файл `formatting.php` содержит функции форматирования WordPress. Они разными способами форматируют данные на выходе:

- `esc_attr` — обрабатывает строку для использования в атрибутах HTML.
- `esc_html` — обрабатывает строку для использования в HTML.
- `esc_url` — используется для проверки и очистки URL.
- `sanitize_text_field` — вычищает строку, введенную пользователем или полученную из базы данных.
- `is_email` — проверяет, действителен ли электронный адрес.
- `capital_P_dangit` — знаменитый фильтр, который букву P в названии WordPress отображает в контенте как строчную.

Pluggable.php

Файл подключаемых функций позволяет вам переписывать определенные функции ядра WordPress. WordPress загружает эти функции, если они все еще не определены после загрузки всех плагинов. Вот некоторые из наиболее широко используемых:

- `wp_mail` — посылает электронные письма из WordPress.
- `get_userdata` — возвращает все пользовательские данные для определенного ID пользователя.
- `get_currentuserinfo` — возвращает все пользовательские данные для пользователя, авторизованного в данный момент.
- `wp_set_password` — обновляет пароль пользователя на новый шифрованный.
- `wp_rand` — генерирует случайное число.
- `wp_logout` — завершает сеанс работы пользователя, прерывая текущую сессию.
- `wp_redirect` — перенаправляет на другую страницу.
- `get_avatar` — возвращает аватар пользователя.

Plugin.php

Файл `plugin.php` содержит функции плагинов API WordPress, в том числе:

- `add_filter` — заставляет ядро WordPress пропускать контент через фильтр, перед тем как отображать его на экране или сохранять в базе данных.
- `add_action` — заставляет ядро WordPress выполнять определенные действия в нужный момент работы.
- `register_activation_hook` — зацепка, вызываемая при активации плагина.

- `register_deactivation_hook` — зацепка, вызываемая при деактивации плагина.
- `plugin_dir_url` — возвращает путь к директории файловой системы для плагина.
- `plugin_dir_path` — возвращает URL для плагина.

User.php

Файл `user.php` содержит функции пользователя API WordPress, в том числе:

- `get_users` — возвращает список пользователей, соответствующих заданным критериям.
- `add_user_meta`, `get_user_meta`, `delete_user_meta` — используется для создания, получения и удаления метаданных пользователя.
- `username_exists` — проверяет, существует ли имя пользователя.
- `email_exists` — проверяет, существует ли электронный адрес.
- `wp_insert_user` и `wp_update_user` — создает и обновляет учетную запись пользователя.

Post.php

Файл `post.php` содержит функции, используемые в обработке записей WordPress, в том числе:

- `wp_insert_post` — создает новую запись.
- `get_posts` — возвращает список последних записей, соответствующих критерию.
- `add_post_meta` — создает метаданные (данные произвольного поля) для записи.
- `get_post_meta` — возвращает метаданные (данные произвольного поля) для записи.
- `get_post_custom` — возвращает многомерный массив со всеми введенными метаданными (произвольные поля) для записи.
- `set_post_thumbnail` — устанавливает миниатюру для записи.
- `register_post_type` — регистрирует индивидуальный тип записи в WordPress.

Функции регистрации плагина `add_filter()` и `add_hook()` — ключи к пониманию того, как WordPress обрабатывает содержание, они позволяют расширить базовые структуры данных контента, используемые WordPress. Мы подробно рассмотрим индивидуальные типы записей и управление данными в главе 7.

Taxonomy.php

Файл `taxonomy.php` содержит функции, используемые таксономиями API WordPress. Таксономии используются для управления иерархическими отношениями

метаданных, такими как категории и метки (см. главу 6) и также могут быть расширены, как вы увидите в главе 7. Среди функций из этого файла:

- `register_taxonomy` — регистрирует пользовательскую таксономию в WordPress.
- `get_taxonomies` — возвращает список зарегистрированных таксономий.
- `wp_insert_term`, `wp_update_term` — вставляет или обновляет элемент таксономии на основе предоставленных аргументов.

Есть куда больше функций ядра, которые могут быть использованы при разработке индивидуальных тем и плагинов для WordPress. Потратьте несколько минут и изучите файлы ядра внутри `/wp-includes`. Эта директория содержит большую часть файлов функций ядра API WordPress.

Чтобы узнать больше о перечисленных здесь функциях, откройте соответствующий файл и просмотрите исходный код. Помните, что каждая функция содержит встроенную документацию, объясняющую, как использовать ее правильно. Мы более подробно рассматриваем функции API плагинов в главе 8. Функции ядра, используемые темами, обсуждаются в главе 9.

Устаревшие функции

С разработкой новой версии WordPress определенные функции могут оказаться устаревшими. «Устаревшая» означает, что функция не удалена из WordPress, но не должна использоваться развивающимися плагинами и темами. Обычно в таком случае создается новая функция для замены устаревшей. Устаревшей она может оказаться по самым разным причинам. Чаще всего дело в том, что она должна быть полностью переписана, чтобы лучше справляться с задачами, которые она выполняет в WordPress.

WordPress содержит файл для хранения всех функций, устаревших с течением времени. Система известна своей превосходной обратной совместимостью. Это значит, что при выходе новой версии WordPress большое внимание уделяется обратной совместимости, чтобы убедиться, что новые особенности и функции не приведут к поломке уже существующих файлов на WordPress, даже если используемые в них особенности будут считаться устаревшими.

Давайте заглянем во встроенную документацию устаревшей функции `get_current_theme()`:

```
/**
 * Возвращает название текущей темы.
 *
 * @since 1.5.0
 * @deprecated 3.4.0
 * @deprecated Use (string) wp_get_theme()
 * @see wp_get_theme()
 *
 * @return string
 */
```

Вы заметите несколько дополнительных строк комментариев для устаревшей функции. Первая — это строка, сообщающая, начиная с какой версии WordPress эта функция считается устаревшей: `@deprecated`, в данном случае с версии 3.4. Вторая — `@see` — сообщает, какую функцию следует использовать взамен, в данном случае `wp_get_theme()`.

Файл `deprecated.php` крайне важно проверять при выходе новой функции WordPress. Если обычная функция устарела, следует немедленно прекратить ее использование и озадачиться обновлением старого кода для использования замены.

В целом устаревшие функции обычно не удаляются из ядра WordPress, но нет никакой гарантии, что этого не произойдет в одной из будущих версий.

Кодекс WordPress и ресурсы

Есть множество различных онлайн-ресурсов, невероятно полезных при изучении WordPress и работы с ним. Эти ресурсы должны быть добавлены в закладки и в равной мере использоваться новичками и экспертами.

Что такое Кодекс?

Кодекс WordPress — это онлайн-вики для документации WordPress, расположенная на WordPress.org. WordPress.org описывает Кодекс как «энциклопедию знаний о WordPress». Вы можете заглянуть туда, зайдя на http://codex.wordpress.org или выбрав вкладку Docs в заголовке WordPress.org.

Кодекс — основанный на вики веб-сайт, это означает, что каждый может создавать, редактировать и добавлять информацию в статьи. Кодекс набит под завязку полезными знаниями по всем аспектам WordPress. От «Начинаем работать с WordPress» до более сложных тем для разработчиков Кодекс является основным ресурсом для тех, что хочет узнать больше о WordPress.

Кодекс доступен на многих языках. Чтобы найти его версию на вашем языке, зайдите на страницу Multilingual Codex http://codex.wordpress.org/Multilingual_Codex. Вы также можете внести свой вклад в Кодекс и помочь расширить его на любом языке или же создать на своем, если этой версии еще не существует.

Использование Кодекса

С Кодексом можно работать разными способами. Наиболее распространенный — использовать в нем поиск с помощью поискового поля в заголовке. Или же вы можете зайти на http://wordpress.org/search/, чтобы найти статьи, соответствующие вашему поисковому запросу.

Поиск WordPress.org поддерживается Google Custom Search (Индивидуальный поиск Google), как показано на рис. 4.1. Все возвращаемые результаты поиска

относятся к WordPress.org, а не только к Кодексу, это важно держать в голове. Есть и менее известный поиск по Кодексу — http://codex.wordpress.org/Special:Search.

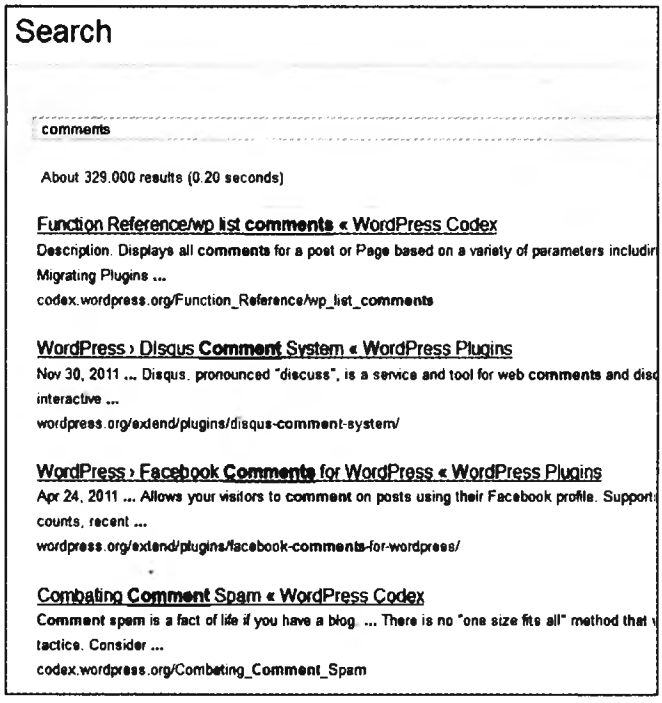


Рис. 4.1. Поиск WordPress.org

Вы также можете перемещаться по индексу статей на домашней странице Кодекса. Эти статьи организуются по темам и обычно сортируются по уровню сложности. Вверху также есть тема по последней версии WordPress. Включенные в нее статьи охватывают новые особенности, тесты на совместимость для плагинов и тем, установку, обновление и поддержку для новой версии.

Для Кодекса существует расширенный глоссарий терминов. Это поможет вам познакомиться со словами, наиболее часто используемыми в Кодексе. Официальный глоссарий Кодекса находится по ссылке http://codex.wordpress.org/Glossary.

Другой метод поиска — использовать быстрый индекс. Он позволяет искать статью по первой букве названия. Быстрый индекс находится по ссылке http://codex.wordpress.org/Codex:Quick_index.

Страница WordPress Lessons (Уроки WordPress) также содержится в http://codex.wordpress.org/WordPress_Lessons.

На этой странице находятся уроки по освоению специфических элементов WordPress. Они организованы по темам. Это прекрасная точка отсчета, если вы не знаете, с чего начать.

Справочник по функциям

Функции WordPress описаны в Кодексе — по отдельной странице для каждой доступной функции API WordPress. На этих страницах подробно объясняется, как именно работает функция WordPress (рис. 4.2). Добавьте этот адрес в закладки для быстрого обращения к информации по функциям WordPress и их возможностям. Официальная страница Справочника по функциям находится по ссылке http://codex.wordpress.org/Function_Reference.

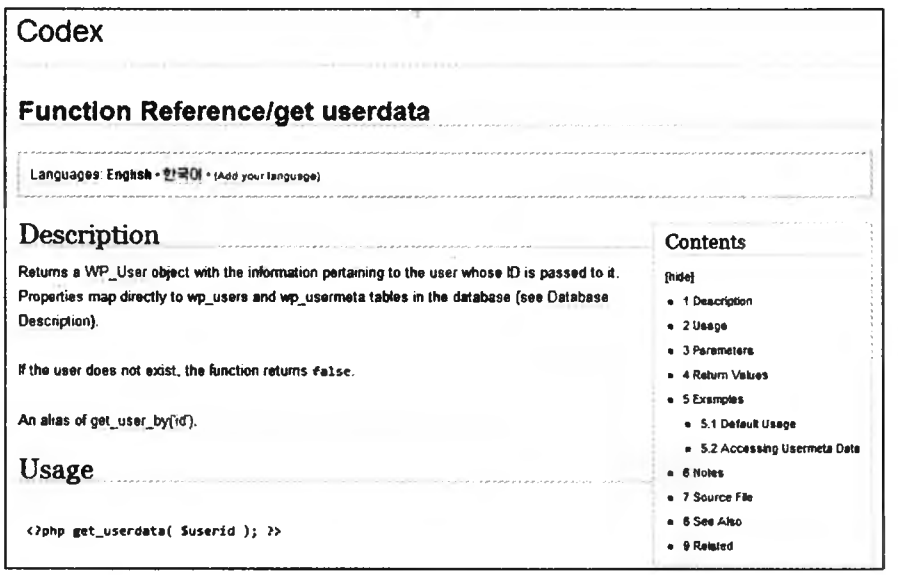


Рис. 4.2. Справочная страница функции get_userdata

Рассматривайте Справочник по функциям как расширенную онлайн-версию встроенной документации функций. В справочнике есть описание работы и использования функции. Индивидуальные параметры перечислены наряду с типами данных и описанием каждого из них.

Самый полезный раздел — примеры — расположен внизу. Примеры позволяют легко увидеть, как использовать функцию. Здесь приведен пример функции `get_userdata`:

```
<?php $user_info = get_userdata(1);
      echo 'Username: ' . $user_info->user_login . "\n";
      echo 'User level: ' . $user_info->user_level . "\n";
      echo 'User ID: ' . $user_info->ID . "\n";
?>
```

Пример показывает, как загрузить конкретные пользовательские данные для пользователя с ID 1. Вариант данных на выходе:

```
Username: michael_myers
User Level: 10
User ID: 1
```

Это простой пример, но наряду с дополнительной справочной информацией он поможет вам изучить новую функцию и понять, как ее правильно использовать в коде.

Последний раздел Справочной страницы содержит список связанных функций. Это поможет вам определить схожие функции, которые могут пригодиться при выполнении задания, находящегося в работе. Например, для функции `wp_insert_post()` в качестве связанных перечислены функции `wp_update_post()` и `wp_delete_post()`.

Большинство функций API WordPress тщательно задокументированы, но не у всех есть Справочные страницы в Кодексе. Любая функция, отображаемая красным на домашней странице Справочника по функциям, на данный момент не снабжена документацией. Это находящийся в постоянной работе проект сообщества, так что ожидается, что когда-нибудь все функции будут полностью описаны в Кодексе.

ЗАМЕЧАНИЕ

Вклад в Кодекс — отличный способ включиться в работу WordPress. Не нужно быть продвинутым разработчиком, чтобы вносить свою лепту в виде примеров кода, описаний и дополнительной информации по возможностям и функциям WordPress.

API WordPress

WordPress содержит множество различных API (интерфейсов прикладных программ), которые помогают взаимодействовать с WordPress. Представьте себе, что API — ворота, которые позволяют вам добавить код или вернуть внешний контент в рамках WordPress без нарушения максимы «не взламывай ядро». Большинство API добавляют отсылки в не относящийся к ядру код, который потом будет добавлен в директорию `wp-content` путем регистрации точек входа WordPress. Каждый API задокументирован в Кодексе наряду с функциями, используемыми в API. API представляет собой набор предопределенных функций, доступных для использования в темах и плагинах. Ниже приведен список наиболее распространенных API WordPress:

- **Plugin API.** Используется для разработки индивидуальных плагинов. В Кодексе есть обширная страница с документацией по Plugin API. Она включает в себя обзор зацепок (перехватывающих функций), действий и фильтров, которые являются основными способами взаимодействия с WordPress с позиции индивидуального плагина. Страница Plugin API ссылается на страницы Справочника по функциям в отношении доступных функций API, которые расположены в `/wp-includes/plugins.php`.

  `http://codex.wordpress.org/Plugin_API`

- **Widgets API.** Используется для создания и поддержания виджета в вашем плагине. Виджет автоматически появится во вкладке **Внешний вид ▸ Виджеты** и может быть размещен на любой заданной панели в теме.

  `http://codex.wordpress.org/Widgets_API`

- **Shortcode API.** Используется для добавления в плагин сокращенных кодов. *Сокращенный код* — это макрокод, добавленный в запись, который позволяет плагину захватить сокращенный код, выполнить определенные команды и отобразить элементы на их месте в записи. Сокращенные коды также могут принимать параметры для изменения данных на выходе.

  Примером сокращенного кода ядра WordPress является `[gallery]`. Добавление `[gallery]` в запись приводит к автоматическому отображению всех изображений, загруженных для этой записи, в виде галереи. При редактировании записи вы будете видеть сокращенный код `[gallery]`, при просмотре страницы на сайте будет отображаться актуальная галерея изображений.

  ```
  http://codex.wordpress.org/Shortcode_API
  ```

- **HTTP API.** Используется для отправки запроса HTTP из WordPress. Этот API является стандартизированным методом захвата контента с внешнего URL. Обычно он берет предоставленный URL и проверяет несколько методов PHP для отправки запроса. В зависимости от среды хостинга WordPress использует первый метод, который полагает правильно настроенным для отправки запроса HTTP.

  Проверенными на данный момент методами HTTP API PHP являются URL, Streams и FSockopen. Методы проверяются именно в этом порядке. Вы можете использовать плагин Core Control (Контроль ядра) (http://wordpress.org/extend/plugins/Core-control/), чтобы специально выбрать, какой метод использовать для отправки запросов HTTP.

  Используя HTTP API, вы можете легко взаимодействовать с Google Maps API, чтобы динамически генерировать карты и планы. HTTP API также может легко взаимодействовать с Twitter API, позволяя вам добавлять/читать твиты прямо из WordPress.

  ```
  http://codex.wordpress.org/HTTP_API
  ```

- **Settings API.** Используется для создания страниц настроек. Этот API используется для создания и управления индивидуальными параметрами ваших плагинов и тем. Основное преимущество использования Settings API — это безопасность. Этот API вычищает все данные настроек, сохраненные пользователем. Это значит, что можно больше не беспокоиться о значении времени, контроле данных и атак через межсайтовый скриптинг (XSS) при сохранении данных по настройкам. Это гораздо проще, чем старый способ контроля данных, который приходилось использовать каждый раз при сохранении настроек плагина.

  ```
  http://codex.wordpress.org/Settings_API
  ```

- **Options API.** Используется для хранения данных по параметрам в базе данных WordPress. Параметры API обеспечивают легкий способ создания, обновления, возврата и удаления значений параметров.

  ```
  http://codex.wordpress.org/Options_API
  ```

- **Dashboard Widgets API.** Используется для создания виджетов администраторской консоли. Виджеты, добавляемые из API автоматически, содержат все возможности jQuery, которые имеют виджеты администраторской консоли, включая параметры перетаскивания, уменьшения и исчезновения с экрана.

  `http://codex.wordpress.org/Dashboard_Widgets_API`

- **Rewrite API.** Используется для создания индивидуальных правил преобразования. Этот API позволяет создавать индивидуальные правила преобразования, как бы вы делали это в файле `.htaccess`. Вы также можете создавать индивидуальные метки структуры постоянных ссылок (то есть `%postname%`), добавлять статические конечные точки (то есть `/my-page/`) и даже дополнительные ссылки для фидов. Функции Rewrite находятся по ссылке `/wp-includes/rewrite.php` в http://codex.wordpress.org/Rewrite_API.

Не забывайте, что все API WordPress могут использоваться при разработке индивидуальных плагинов и тем. Это основной метод расширения WordPress с помощью дополнительных возможностей и функциональности. Использование названных выше API дает простой и стандартизированный путь взаимодействия с WordPress.

Для получения дополнительной информации по всем API WordPress обратитесь к странице Кодекса http://codex.wordpress.org/WordPress_API's.

Битва за Кодекс

Как и в случае любой вики, споры о точности статей в Кодексе будут вестись всегда. Одна из бед, поразивших Кодекс, — это актуальность статей. WordPress идет вперед бодрой рысью, которой должен бежать и Кодекс, чтобы оставаться точным. К сожалению, он поспевает не всегда, поэтому некоторые материалы оказываются устаревшими. Кодекс WordPress — коллективный проект, поэтому вы можете без проблем создать в нем учетную запись и оказывать посильную помощь.

Другой сложностью Кодекса является организация содержимого. На данный момент море информации раскинулось настолько широко, что выловить из него ответы на нужные вопросы может быть ой как непросто. Так что, как уже говорилось, введение в ядро WordPress снабдило вас картой, позволяющей сузить область поиска, и познакомило вас с темами, касающимися функций.

Не взламывайте ядро!

Хотя исследование ядра WordPress и использование его в качестве справочника в высшей степени вдохновляет, не надо взламывать ядро. Взломать ядро — значит внести какие-либо изменения в файлы ядра WordPress. Изменение может касаться всего лишь одной строки кода, однако взлом есть взлом, и он способен вызвать огромные проблемы в дальнейшем.

Почему нет?

Взлом ядра WordPress может до крайней степени усложнить обновление до последней версии WordPress. Поддерживать WordPress в актуальном состоянии — важный шаг на пути к общей безопасности всего веб-сайта. При обнаружении любой уязвимости патч обычно выпускается очень быстро. Если вы не можете обновить систему, потому что изменили файлы ядра, вы оставляете эти уязвимости открытыми, тем самым увеличивая вероятность взлома сайта.

Взлом ядра также может привести к нестабильности сайта, поскольку многие компоненты WordPress функционируют в прямой зависимости от правильной работы других компонентов. Если вы вносите изменения в такие компоненты, сломанным может оказаться нечто, совершенно не связанное с точкой внесения изменений.

Безопасность — еще одна причина, по которой не следует взламывать ядро. Ядро WordPress пристально изучается специалистами по безопасности со всего мира. Взламывая ядро, вы полагаетесь на собственные знания в вопросах дальнейшей безопасности. Если вы не понимаете разнообразия путей, которыми хакер может взломать ваш код, то рискуете закончить созданием уязвимости в ядре WordPress.

Наконец, последняя причина, по которой вам не следует взламывать ядро, — это сочувствие. Сочувствие разработчикам, которым придется поддерживать сайт после вас. Разработчики могут меняться, поэтому нет гарантии, что вы будете работать над этим сайтом через пять лет. Представьте разработчика, который вслед за вами пытается определить, какие файлы ядра были взломаны, чтобы заставить веб-сайт работать. Это может оказаться кошмаром для любого и поставить владельца сайта в дурацкое положение, поскольку большинство разработчиков будут отказываться работать со взломанной версией WordPress. Взламывая код, вы создаете зависимости, которые будут либо неверно поняты, либо скрыты, и при обновлении сайта взломанное ядро рванет — тихо ли, громко ли.

Альтернативы взламыванию ядра

Любые возможности или функциональность, не существующие в WordPress, могут быть добавлены с помощью плагина. Иногда взлом ядра может быть простым ответом, но в долгосрочной перспективе это усложнит вам жизнь. (Мы еще не дошли до возможностей, которые нам нужны, но не могут быть внедрены с помощью плагина.) WordPress — невероятно гибкая система, что является одной из ее сильных сторон, поэтому ядро никогда не следует взламывать. Не взламывайте ядро!

Если вы очарованы ядром WordPress и его лабиринтами, присоединяйтесь к сообществу разработчиков WordPress, участвуйте в устранении ошибок и вносите свой вклад в развитие ядра.

Резюме

В этой главе мы рассмотрели ядро программного обеспечения WordPress. Мы выяснили, что содержится в ядре, как использовать его в качестве справочника при разработке WordPress и как определить, какие функции устарели при выпуске очередной версии. Мы также познакомились с Кодексом WordPress и доступными в WordPress API.

Теперь, когда вы понимаете, что такое ядро WordPress, пора выяснить, как использовать цикл WordPress для отображения контента согласно вашим нуждам.

5

Цикл (Loop)

В этой главе:

- Понимание работы цикла и возможностей его использования
- Определение отображения контента с использованием цикла
- Индивидуализация цикла за счет разной степени доступа к данным
- Использование меток шаблонов
- Понимание глобальных переменных и их соотношения с обработкой цикла
- Работа вне цикла

Цикл имеет отношение к тому, как WordPress отображает контент (записи, страницы или пользовательский контент) на посещаемой странице. Цикл может выдавать отдельный фрагмент содержимого или группу записей или страниц, которые выбраны и отображаются посредством кругового обхода, поэтому процесс и называется циклом.

Как WordPress отображает записи блога по умолчанию? Цикл выбирает записи из базы данных MySQL на основе набора параметров. Параметры обычно определяются через URL, используемый для доступа к сайту WordPress. Например, домашняя страница по умолчанию должна отображать все записи блога в обратном хронологическом порядке. Страница категории, на которую можно попасть, используя URL типа http://example.com/category/zombies, показывает только записи блога, относящиеся к этой категории (в данном случае к категории «зомби»). Страница архива показывает только записи блога, датированные определенным месяцем и датой. WordPress превращает практически каждый параметр записи в выбранную переменную, давая циклу возможность использования разных

алгоритмов выбора контента. Настроить тип и место отображения контента на сайте легко, если понимаешь, как цикл превращает URL в то, что отображается при переходе по ссылке.

В этой главе обсуждается, как работает цикл, где его можно использовать и его логическая схема. Здесь же описываются возможности индивидуальной настройки цикла с использованием множества различных функций и методов доступа к данным, имеющихся в WordPress. Речь пойдет еще и о глобальных переменных, поддерживающих текущее состояние, и о работе вне цикла.

Понимание цикла

Понимание того, как функционирует цикл, поможет вам им управлять. Управление циклом с целью отображения именно того контента, который вам нужен, будет одним из самых часто используемых инструментов при разработке сайтов на WordPress. Поскольку цикл — в сердце каждой темы WordPress, возможность настроить отображаемый контент таким образом, чтобы заставить WordPress выглядеть и действовать так, как вам хотелось бы.

Чтобы понять цикл, полезно разобрать шаги, которые WordPress предпринимает, чтобы сгенерировать контент страницы:

1. URL соответствует файлам и директориям, имеющимся в копии WordPress. Если файл существует, он загружается веб-сервером. WordPress, по сути, не участвует в этом решении. Должен ли URL обрабатываться веб-сервером или же его следует превратить в запрос по контенту WordPress, решают веб-сервер и созданный WordPress файл `.htaccess`. Об этом говорилось в главе 2 при изучении постоянных ссылок.

2. Если URL не загружает файл ядра WordPress, он должен быть проанализирован, чтобы решить, какой контент загружать. Веб-сервер начинает с загрузки ядра WordPress через `index.php`, чтобы задать параметры для цикла. При посещении страницы какой-либо метки, например http://example.com/tag/bacon, WordPress определит, что вы просматриваете метку, загрузит соответствующий шаблон, выберет записи, сохраненные с этой меткой, и сгенерирует выходные данные для страницы с меткой.

3. Магическая трансформация URL в выбранный контент происходит внутри метода `parse_query()` в объекте `WP_Query`, который WordPress создал ранее при обработке. WordPress сначала разбирает URL на набор параметров запроса, которые описаны в следующем разделе. Все строки запроса из URL передаются для определения отображаемого контента, даже если они выглядят как аккуратно отформатированные пути. Если ваш сайт использует «красивые» постоянные ссылки, значения между слэшами — просто параметры для строк запроса. Например, http://example.com/tag/bacon — то же самое, что и http://example.com?tag=bacon, передающая строку запроса тега со значением bacon.

4. После этого WordPress конвертирует детальные параметры запроса в запрос для базы данных MySQL, чтобы вернуть контент. Рабочей лошадкой тут является метод `get_posts()` внутри объекта `WP_Query`, который описывается в этой части далее. Метод `get_posts()` берет все параметры запроса и превращает их в операторы SQL, в конечном счете посылая строку SQL на сервер баз данных MySQL и извлекая желаемый контент. Контент, возвращенный из базы данных, сохраняется в объекте `WP_Query`, чтобы использоваться в цикле WordPress, и кэшируется для ускорения других обращений к тем же самым записям до выполнения другого запроса базы данных.
5. После возвращения контента WordPress устанавливает все условные метки `is_`, такие как `is_home()` и `is_page()`. Они устанавливаются как часть выполняемого запроса по умолчанию на основе парсинга URL, и мы еще рассмотрим случаи, когда может потребоваться сбросить эти метки.
6. WordPress берет образец из темы, основываясь на типе запроса и числе возвращенных записей, — например, единичная запись или запрос по категории, — и полученные по запросу данные передаются в процедуру цикла по умолчанию.

Цикл можно настроить для различных целей. Например, новостной сайт может использовать цикл для отображения заголовков последних новостей. Деловой каталог может применять цикл для отображения локальных предпринимателей в алфавитном порядке или всегда помещать записи о фирмах-спонсорах в верхнюю часть каждой отображаемой страницы. Электронный магазин может использовать цикл для отображения продукции, загруженной на сайт. Возможности настройки цикла бесконечны, поскольку он дает вам полный контроль над выбором контента и порядком его отображения на экране.

От параметров запроса к SQL

После того как заданы параметры запроса — посредством парсинга URL, заданного пользователем, или явного задания в специально настроенном цикле, — метод `get_posts()` объекта `WP_Query` транслирует эти параметры в SQL для запроса базы данных. Помимо того что вы можете во многом контролировать тип, выбор и упорядочивание контента с помощью параметров запроса, ядро WordPress также предоставляет фильтры, чтобы позволить вам изменить сгенерированный SQL для тончайшего контроля выбора контента и его группировки.

Базовый формат запроса SQL: `SELECT fields FROM table WHERE conditions` (ВЫБРАТЬ *поля* ИЗ *таблицы* ГДЕ *условия*). *Поля* — это столбцы базы данных, которые вы хотите вернуть. Обычно эту часть запроса менять не требуется. *Условия*, определенные в операторе WHERE, меняют порядок, группировку и число возвращаемых записей. Если вы выгрузите сгенерированный запрос SQL, проанализировав поле `request` объекта `WP_Query`, то увидите, что `WHERE` в SQL содержит `1=1` как первое условие. Если нет других параметров выбора контента, `1=1` гарантирует, что сгенерированный SQL не будет синтаксически плохо сформированным в отсутствие

других выражений в `WHERE`. Оптимизатор SQL в MySQL знает достаточно, чтобы проигнорировать `1=1`.

Таблица — это не просто «записи» таблицы в базе данных MySQL, которые содержат все данные записей. Это также может быть SQL JOIN из двух или большего числа таблиц, где вам необходимо выбрать записи на основе иерархических метаданных. В WordPress любая запись может быть легко отмечена разными метками или помещена в более чем одну категорию. Но соответствующие базы данных не приспособлены для управления этими иерархическими или сетевыми отношениями. Как вы увидите в главе 6, модель данных WordPress использует множественные таблицы для управления этими сложными взаимоотношениями, но это делает такие запросы, как «найти все записи с меткой "bacon"», более сложными для выполнения. Например, чтобы выбрать записи с меткой `bacon`, SQL JOIN необходимо сначала найти `bacon` в таксономии метаданных, построить в памяти промежуточную таблицу записей, которые были отмечены как `bacon`, а затем выбрать записи, ID которых содержатся одновременно и во временной таблице, и в главной таблице контента WordPress. Приверженцы баз данных называют это «картезианским продуктом», или внутренним слиянием двух и более таблиц. Мультипликативное описание, сложность запроса и потребление памяти выражены здесь очень точно.

В главе 8 мы будем подробнее разбирать плагины, их применение к фильтру и действие зацепок, связанных с ядром WordPress. При генерации запроса SQL есть некоторое число фильтров, которые запускаются, чтобы обеспечить авторам плагина динамическое связывание и в высшей степени ясный контроль над выполняемым SQL. Например, возьмем плагин, который меняет выборку записей в зависимости от пользовательских метаданных записи и контекста, которые плагин держит в отдельной таблице базы данных. Ваш плагин будет использовать фильтр `posts_join`, чтобы перезаписать оператор `JOIN`, добавляя еще одну таблицу и поле, соответствующее оператору, для дальнейшего расширения набора выбора. Если вы хотите изучить ядро на предмет наличия «кровавых» подробностей генерирования SQL, большая часть парсинга запроса (базы данных) в запрос выполняется в `wp-includes/query.php`, а большая часть работы `JOIN` определяется в `wp-includes/taxonomy.php`.

И еще одно последнее замечание по генерированию SQL. WordPress выполняет отличную работу по построению канонических URL, то есть это один-единственный способ ссылки на конкретную запись. Поисковые системы неизменно считают разными страницами http://example.com/bacon и http://example.com/2012/bacon, даже если они отображают один и тот же фрагмент контента (это сделано по большей части, чтобы помешать общеизвестной практике *фарминга ссылок* (*link farming*), при которой ссылки генерируются для повышения популярности одной цели). Часть функций парсинга URL внутри ядра WordPress пытаются очистить URL и привести их к канонической форме. Те же функции прилагают все усилия к возврату какой-то информации, а не страницы 404. В результате, если попытка загрузить страницу по имени не дает какого-либо контента, WordPress добавляет модификатор `LIKE` в оператор `WHERE`, который содержит имя записи. Например, если пользователь вводит http://example.com/2012/scott, но у вас нет записей Scott, оператор `LIKE` выдаст

все записи, начинающиеся со Scott, например *Scott Gomez Finally Scored*. Канонические URL и соответствие «подобных имен» являются частью сложного лабиринта преобразования URL и направленного парсинга, который пытается сгенерировать нечто приятное для пользователя, а не надоевшую ошибку 404.

Понимание контента в WordPress

Перед тем как погружаться в Loop, важно понять, какие типы контента существуют в WordPress. По умолчанию WordPress определяет два типа содержимого: записи и страницы. В главе 6 вы увидите, что все типы содержимого хранятся в одной таблице MySQL и различаются по «типу». Начиная с версии WordPress 2.9 можно определять пользовательские типы записей, по сути являющиеся пользовательским контентом WordPress. Например, вы можете создать пользовательский тип записи «События», чтобы регистрировать события в WordPress.

В этой главе под контентом подразумеваются «записи», но важно помнить, что записи на самом деле могут быть контентом WordPress любого типа. Некоторые типы записей рассматриваются в главе 7.

Помещение цикла в контекст

Цикл — это сердце темы, контролирующее отображение контента. Это функциональное соединение между базой данных MySQL и HTML, которое генерируется в браузере посетителя. В целом, где бы ни отображались запись или страница, WordPress использует цикл. Это может быть одиночная запись или страница, цикл записей или последовательность циклов с различными параметрами отображения.

Большинство тем WordPress включают в себя заголовок, подвал и боковые панели. На рис. 5.1 показано, как цикл помещается прямо посредине этих элементов, создавая область контента сайта. Эта часть вашего сайта обычно является динамической и будет меняться по мере навигации через нее.

Цикл по умолчанию используется в файлах шаблона вашей темы WordPress. Цикл можно создавать где угодно в файлах шаблона темы, как показано на рис. 5.2. Цикл с индивидуальными настройками используется в плагинах и виджетах. Его можно использовать где угодно внутри WordPress, но в зависимости от конкретного места существуют разные методы создания цикла с индивидуальной настройкой, а потенциальные побочные эффекты у каждой конструкции свои.

В файлах шаблона темы цикл можно использовать многократно. Циклы могут создаваться в заголовке, на боковых панелях, в подвале и в области основного контента сайта. Количество циклов, которые могут отображаться на сайте, не ограничено. Не забывайте, что цикл, по сути, — запрос базы данных для выбора контента и последующего цикла для обработки и отображения последнего. По умолчанию для осуществления выборки цикл использует контекст запрошенного URL, но вы можете провести тонкую настройку и создать запрос для базы контента WordPress, чтобы выполнять любое число процессов управления содержимым.

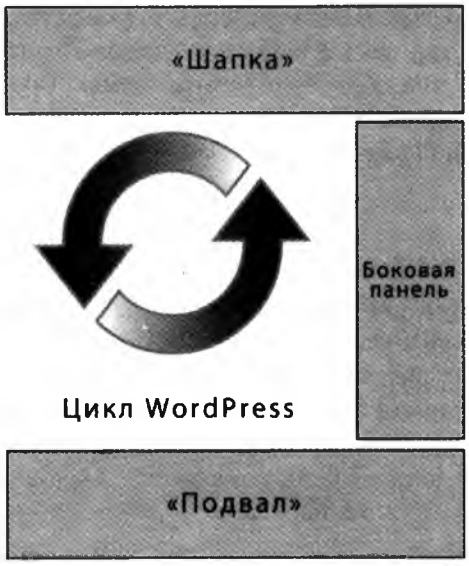


Рис. 5.1. Цикл WordPress

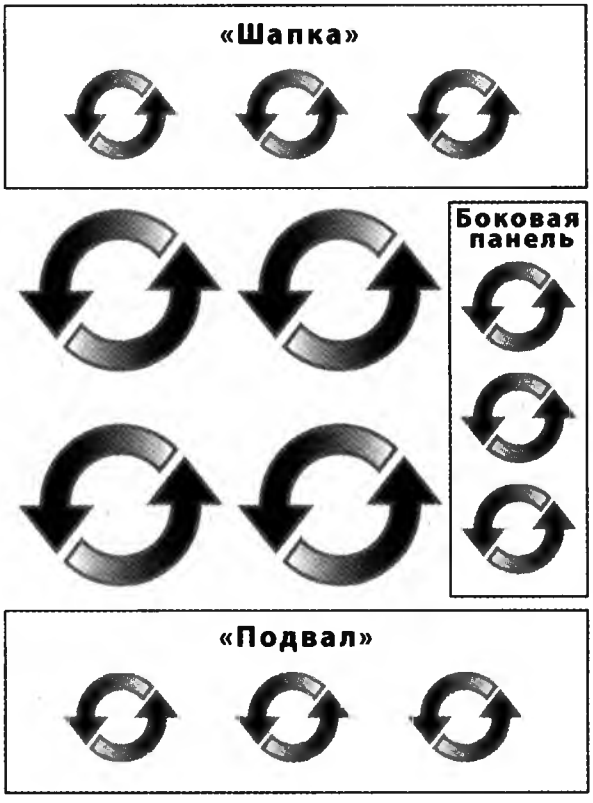


Рис. 5.2. Использование нескольких циклов

В следующем разделе рассматривается контроль базового процесса цикла и функции шаблона WordPress, предоставленные для индивидуальной настройки способа отображения контента при обработке внутри цикла. Вооружившись основами, теперь вы займетесь построением пользовательских циклов на основе самостоятельного формирования параметров запросов.

Процесс цикла

Loop использует некоторые стандартные для программирования условные операторы, чтобы определить, что и как отображать. Первым оператором в Loop является `if`, который проверяет, существует ли хотя бы одна запись, поскольку у вас может не быть записей в выбранной категории и с нужной меткой. Если контент существует, оператор `while` используется для запуска цикла, охватывающего все записи или страницы, которые должны быть отображены. Наконец, функция `the_post()` вызывается для построения данных записи, делая их доступными для других функций WordPress. После построения данных записи контент цикла может быть отображен в том формате, который вы предпочитаете.

Ниже приведен минимальный пример цикла. Он включает в себя элементы, необходимые для правильного функционирования цикла:

```
<?php
if ( have_posts() ) :
   while ( have_posts() ) :
      the_post();
      // код внутри цикла (теги шаблонов, html и пр.)
   endwhile;
endif;
?>
```

Помните, что это код PHP, поэтому он должен быть заключен в теги `<?php` и `?>`. Это цикл в простейшем варианте. Если вы гадаете, как информация на выходе запроса базы данных передается этому простейшему Loop, в котором нет переменных в качестве параметров, ответ лежит в глобальной переменной `$wp_query`, являющейся экземпляром `WP_Query`, на которую ссылаются функции в простом цикле. По сути, это запрос для цикла по умолчанию. Заметьте, что к моменту вызова этого цикла по умолчанию WordPress уже вызвал метод `get_posts()` в рамках объекта запроса по умолчанию, чтобы построить список искомого контента для просматриваемого URL, а цикл в данном случае отвечает за отображение этого списка записей. Позже вы увидите, как вручную структурировать запросы, чтобы осуществлять высокоточный контроль над выбором записей, но на данный момент безопаснее положиться на четкую работу базы данных, после чего результаты при вызове цикла сохраняются в `$wp_query`.

Существуют минимальные требования для работы цикла в WordPress. Давайте разберем пример, чтобы изучить отдельные части цикла:

```
if ( have_posts() ) :
```

Эта строка проверяет, будут ли на текущей просматриваемой странице отображаться какие-либо записи и страницы. Если записи и страницы существуют, будет выполняться следующее предложение:

```
while ( have_posts() ) :
```

Содержащийся выше оператор `while` запускает цикл, обыкновенно проходя через все записи и страницы, которые должны быть отображены на странице, если не задано ничего другого. Цикл будет продолжаться, пока есть контент для отображения. Когда все содержимое будет отображено, цикл закончится. Функция `have_posts()` просто проверяет, полностью обработан список записей или же в списке нет записей.

```
the_post();
```

Теперь вызывается функция `the_post()` для загрузки всех данных записи. Эта функция должна быть вызвана внутри цикла, чтобы данные записи были установлены корректно. Вызов `the_post()` в свою очередь вызывает функцию `setup_postdata()`, чтобы установить метаданные записи, такие как автор и метки контента, который вы отображаете в цикле, а также содержимое самой записи. Эти данные передаются глобальной переменной каждый раз через повтор цикла. У специально вызываемого `the_post()` есть побочный эффект в виде установки глобальной переменной `$post`, используемой большинством описанных далее шаблонов, которая затем передается следующей записи в списке.

Установка данных записи также использует соответствующие фильтры для сырого контента, который выходит из базы данных WordPress. WordPress хранит отредактированное пользователем содержимое в том виде, в каком оно было введено. Поэтому если пользователь, к примеру, использовал сокращенный код, чтобы добавить объект Google AdSense в конце записи, этот код будет храниться в контенте базы данных. Когда настройка записи завершена, вызывается плагин, который конвертирует сокращенные коды в куски JavaScript наряду с другими зарегистрированными плагинами, которые модифицируют контент сырой записи. Вы посмотрите на механику плагинов в главе 8, но сейчас важно заметить разницу между сырыми данными записи в объекте запроса WordPress и отфильтрованным контентом, который формируется в конце.

```
// код внутри цикла
```

Сюда помещаются теги всех шаблонов цикла и любой дополнительный код, который вы хотели бы отобразить внутри цикла. Эта тема будет более подробно рассмотрена далее.

```
endwhile;
endif;
```

Операторы `endwhile` и `endif` заканчивают цикл. Любой код, помещенный после этих двух строк, будет отображаться внизу страницы после всех записей. Вы также можете поместить оператор `else` для отображения сообщения, если содержания для отображения в Loop нет.

В файлах шаблона темы цикл обычно окружен тегами HTML. Приведенный ниже код показывает, как цикл структурирован в теме Twenty Eleven, поставляемой вместе с WordPress:

```
<div id="primary">
    <div id="content" role="main">
    <?php if ( have_posts() ) : ?>
        <?php twentyeleven_content_nav( 'nav-above' ); ?>
        <?php /* Начинается цикл */ ?>
        <?php while ( have_posts() ) : the_post(); ?>
            <?php get_template_part( 'content', get_post_format() ); ?>
        <?php endwhile; ?>
        <?php twentyeleven_content_nav( 'nav-below' ); ?>
    <?php else : ?>
        <article id="post-0" class="post no-results not-found">
            <header class="entry-header">
                <h1 class="entry-title">
                    <?php _e( 'Nothing Found', 'twentyeleven' ); ?>
                </h1>
            </header><!-- .entry-header -->
            <div class="entry-content">
                <p>
                    <?php _e( 'Apologies, but no results were found
                    for the requested archive. Perhaps searching will
                    help find a related post.', 'twentyeleven' ); ?>
                </p>
                <?php get_search_form(); ?>
            </div><!-- .entry-content -->
        </article><!-- #post-0 -->
    <?php endif; ?>
    </div><!-- #content -->
</div><!-- #primary -->
```

Заметьте, как существуют минимальные элементы цикла, окруженные тегами HTML. Именно так структурируется нормальный файл шаблона темы для использования цикла. Элементы HTML, разумеется, могут меняться, но элементы цикла остаются именно такими. Настройка стиля отображения контента и выбор метаданных записи, которые будут отображаться на странице, осуществляются через теги шаблона.

Теги шаблона

Функции PHP, используемые шаблонами тем WordPress для отображения контента цикла, называются *тегами шаблона* (*template tags*). Эти теги используются для отображения тех или иных фрагментов данных вашего сайта и контента. Это позволяет вам настроить отображение контента на сайте.

Например, тег шаблона `the_title()` отображает название записи или страницы внутри Loop. Основное преимущество использования тегов шаблона в том, что вам не нужно знать код PHP для работы с ними.

На WordPress доступно много различных шаблонов. Некоторые теги шаблонов должны находиться внутри цикла, тогда как другие могут использоваться где

угодно в файлах шаблона. Заметьте, что в этом контексте под тегами шаблона понимаются функции WordPress, используемые для извлечения данных записи для отображения. Файлы шаблона — это элементы темы, которые контролируют, как отображается тот или иной тип контента. Иначе говоря, файлы шаблона содержат циклы, включающие в себя теги шаблона. Обновляемый список тегов шаблона доступен по ссылке http://codex.wordpress.org/Template_Tags.

Часто используемые теги шаблона

Тегов шаблона существует много, но обычно в цикле вы видите только их небольшую часть. Вот наиболее часто используемые теги шаблона, доступные в цикле. Эти теги шаблона будут возвращать и отображать перечисленные данные записи.

- `the_permalink()` — отображает URL записи.
- `the_title()` — отображает название записи.
- `the_ID()` — отображает уникальный ID записи.
- `the_content()` — отображает полное содержание записи.
- `the_excerpt()` — отображает цитату записи. Будет использоваться, если заполнено поле **Цитата (Excerpt)** на экране редактирования записи. Если нет, то WordPress сгенерирует отрывок автоматически.
- `the_time()` — отображает дату/время публикации записи.
- `the_author()` — отображает автора записи.
- `the_tags()` — отображает метки записи.
- `the_category()` — отображает категории записи.
- `edit_post_link()` — отображает ссылку **edit**, которая показывается, только если вы авторизованы. Она позволяет редактировать запись.
- `comments_popup_link()` — отображает ссылку на форму комментария к записи.

Чтобы понять, как работают теги шаблона, просто поместите любой тег внутрь Loop и посмотрите, что получится. Следующий пример показывает, как работает пара таких тегов:

```
<?php
if ( have_posts() ) :
   while ( have_posts() ) :
      the_post();
      ?>
      <a href="<?php the_permalink(); ?>"><?php the_title(); ?></a>
      <br />
      <?php
      the_content();
   endwhile;
endif;
?>
```

Как видите, названия записей отображаются с постоянными ссылками на каждую запись. Контент записи отображается прямо под названием.

Параметры тегов

У большинства тегов шаблона есть параметры, которые могут быть добавлены для изменения возвращаемого значения. Например, у тега `the_content()` есть два параметра. Первый позволяет устанавливать ссылку more на дальнейший текст:

```
<?php the_content( 'Read more', false ); ?>
```

Контент записи будет отображаться нормальным образом, но при наличии в записи тега `<!--more-->` WordPress будет автоматически добавлять текст `Read more`, который будет снабжен ссылкой на полную запись. Второй параметр определяет, следует ли отображать параграф-тизер еще раз при просмотре полной записи. По умолчанию стоит значение `false`, и тизер отображается в обоих местах.

ЗАМЕЧАНИЕ

Тег `more` в WordPress позволяет вам отображать определенный тизер из полной записи на сайте. Например, вы можете показать первый параграф записи на первой странице, а полная запись будет отображаться, только если посетитель перейдет по соответствующей ссылке. Чтобы сделать это, поместите `<!--more-->` в контенте в виде HTML в месте, где хотите вставить разрыв. В визуальном редакторе есть кнопка `More` для вставки этого тега.

Вы также можете присвоить несколько параметров любому тегу шаблона, который их поддерживает. Например, тег шаблона `the_title()` принимает три параметра: `$before`, `$after` и `$echo`. Следующий код задает тегу `the_title()` параметры `$before` и `$after` для помещения названия записи в теги `h1`:

```
<?php the_title( '<h1>', '</h1>' ); ?>
```

Вы также можете просмотреть актуальные функции в исходном коде WordPress. Функции шаблона записи находятся в `wp-includes/post-template.php`. Осуществляя быстрый поиск `function the_title()`, вы найдете точное описание функции для тега `the_title()`. Вы также можете использовать Кодекс для получения подробного описания тега шаблона, с которым работаете. В данном случае: http://codex.wordpress.org/Template_Tags/the_title.

Индивидуальная настройка цикла

В обсуждении процесса контроля цикла было замечено, что главной рабочей лошадкой для выбора данных является метод `get_posts()` объекта `WP_Query`. В большинстве случаев, если вы хотите построить пользовательский цикл, вы построите свой собственный `WP_Query` и будете ссылаться на него. В качестве альтернативы вы можете использовать функции нижнего уровня `query_posts()` и `get_posts()`

(не путайте с методами внутри объекта `WP_Query` с теми же названиями), чтобы управлять данными на выходе запроса по умолчанию, прошедшими через цикл. И `query_posts()`, и `get_posts()` используют класс `WP_Query`, чтобы возвращать содержимое. Мы рассмотрим подход нижнего уровня и обсудим, как и где его следует — и не следует — использовать, но начать нужно с разговора о том, как построить произвольный объект запросов.

Использование объекта WP_Query

Когда WordPress получает URL для парсинга от веб-сервера, он разбирает знаки в этом URL и конвертирует их в параметры для запроса базы данных. Вот несколько более детальное описание того, что происходит при управлении экземпляром `WP_Query`.

`WP_Query` — это класс, определенный в WordPress, который облегчает создание собственных циклов. И `query_posts()`, и `get_posts()` используют класс `WP_Query` для возврата контента WordPress. Когда вы используете `query_posts()`, глобальная переменная `$wp_query` используется как копия `WP_Query`, делая `$wp_query` хранилищем данных по умолчанию для нескольких операций. Произвольные циклы могут использоваться где угодно в файлах шаблона темы для отображения различных типов содержимого. Они должны создавать отдельные экземпляры переменной `WP_Query`.

Когда вы создаете новый объект `WP_Query`, он обладает несколькими предустановленными функциями для построения запросов, выполнения запросов для получения записей и парсинга параметров из URL. Однако вы можете использовать эти встроенные объектные методы для построения собственных строк параметров и создания произвольных циклов, которые извлекают любой необходимый контент, указанный в цикле.

Ниже приведен пример произвольного цикла, отображающего пять последних записей на сайте:

```
<?php
$myPosts = new WP_Query( 'posts_per_page=5' );while ( $myPosts->have_posts() )
: $myPosts->the_post();
?>
  <!-- делаем, что нам нужно -->
<?php endwhile; ?>
```

Вместо использования несложных вызовов `have_posts()` и `the_post()`, которые вы видели в базовом цикле, произвольный цикл вызывает методы заново созданного объекта WP_Query `$myPosts`. Явный вызов процедуры, показанный здесь, и вызов по умолчанию `have_posts()` функционально эквиваленты; `have_posts()`, например, представляет собой просто вызов `$wp_query->have_posts()` с использованием глобальной переменной запроса для запроса по умолчанию, то есть той, которая сгенерирована из парсинга URL, переданного WordPress веб-сервером.

Переход к циклу по умолчанию от URL вызывает WordPress. Есть дополнительный шаг, который состоит в том, что осуществляется парсинг URL для получения

нужной строки запроса с использованием метода `parse_query()` объекта запроса. При построении произвольного цикла вы подробно задаете параметры, контролирующие запрос. Вот более детальная информация о том, что происходит внутри функции запроса:

- Вызов `$myPosts->query()` конвертирует параметры в SQL-операторы посредством функции `$myPosts->get_posts()`, которая затем выполняет запрос к базе данных MySQL и извлекает запрошенный контент.
- Что не менее важно, вызов запроса устанавливает условные теги, такие как `is_home()` и `is_single()`, которые зависят от типа отображаемой страницы и количества контента для нее.
- Массив записей, возвращаемый запросом, кэшируется WordPress, чтобы будущие обращения с тем же запросом не генерировали дополнительно трафик для базы данных.

Чтобы строить мощные произвольные циклы, необходимо правильно трансформировать критерии выбора контента в набор параметров запроса.

Построение произвольного запроса

Параметры используются для определения того, какой контент будет возвращен в цикле, неважно, произвольный это цикл или базовый. При создании цикла важно понимать, какие параметры доступны, чтобы помочь определить контент, который будет отображен. Вы можете использовать множество разных, иногда противоречивых, параметров при создании произвольного цикла для настройки контента на выходе.

Множественные параметры также могут быть выставлены через запрос посредством разделения имени параметра и значений амперсандом. Для получения более подробного списка доступных параметров загляните по ссылке http://codex.wordpress.org/Class_Reference/WP_Query#Parameters.

В следующих разделах рассматриваются некоторые из наиболее часто используемых параметров.

Параметры записи

Наиболее очевидные и порой наиболее часто используемые параметры определяют число и тип отображаемых записей:

- `p=2` — загрузка отдельной записи по ID.
- `name=my-slug` — загрузка записи на основе слага (slug) записи («хвоста» постоянной ссылки).
- `post_status=pending` — загружает записи по статусу. Например, если вы хотите просмотреть только черновики, выберите `post_status=draft`.

- `ignore_sticky_posts` — исключает возвращение приклеенных записей первыми. *Приклеенная запись* (sticky post) — та, которая всегда оказывается сверху в списке записей, вне зависимости от других параметров, выставленных для запроса. Вы можете сделать несколько «приклеенных» записей, чтобы привлекать внимание к новостям, акцентировать изменения или иным образом удерживать внимание читателя. Этот параметр позволит убрать такие записи из приоритетных в верхней части списка.
- `post_type=page` — загружает записи по типу. Если вы хотите просматривать только страницы, не записи, это поможет вам возвращать их: `post_type=page`. Данный параметр активирует специальные циклы для отбора контента на основе типа записей, как вы увидите в главе 7.
- `posts_per_page=5` — число записей для загрузки на страницу. Значение по умолчанию. Чтобы показывать все записи, выставьте значение параметра –1.
- `offset=1` — число записей, пропускаемых до загрузки.

Параметры страницы

Среди параметров страниц есть схожие с параметрами записей для контроля их отбора:

- `page_id=5` — загрузка отдельной страницы по ID. Как и ID записей или пользователей, ID страницы можно найти в Консоли, наведя курсор на страницу и посмотрев на URL, отображаемые внизу окна браузера.
- `pagename=Contact` — загружает страницу по названию, в данном случае — Contact.
- `pagename=parent/child` — загружает дочернюю страницу по слагу или иерархии слагов (по пути).

Параметры категорий, меток и авторов

Записи также могут быть отсортированы по категории, в которую они были помещены, по меткам или по информации об авторе:

- `cat=3,4,5` — загружает страницы по ID категории.
- `category_name=About Us` — загружает страницы по имени категории. Обратите внимание, что если запись помещена более чем в одну категорию, она будет отображаться в выборке по каждой из них.
- `tag=writing` — загружает записи по имени метки.
- `tag_id=34` — загружает записи по ID метки.
- `author=1` — загружает записи по ID автора.
- `author_name=brad` — загружает записи по имени автора.

Параметры времени, даты, упорядочивания и произвольные параметры

Параметры для выбора контента на основе хронологии — ключевая часть построения архива записей или просмотра содержимого через календарь на домашней странице блога. Вы также можете изменить параметры и порядок сортировки. Если вы создаете онлайн-индекс и хотите показать список записей по алфавиту, установите параметры для запроса записей по месяцу и автору, но упорядочите результаты по названию.

- `monthnum=6` — загружает записи, созданные в июне.
- `day=9` — загружает записи, созданные в 9-й день месяца.
- `year=2012` — загружает записи, созданные в 2012 году.
- `orderby=title` — поле для по сортировки записей по...
- `order=ASC` — определяет возрастающий или убывающий порядок `orderby`.
- `meta_key=color` — загружает записи по имени произвольного поля. Обратитесь к обсуждению пользовательской таксономии в главе 7, чтобы увидеть, как добавлять в записи произвольные поля.
- `meta_value=blue` — загружает записи по значению произвольного поля. Может быть использован вместе с приведенным выше параметром `meta_key`.

А теперь все вместе

Теперь рассмотрим некоторые примеры использования параметров. Следующие примеры используют функцию `$myPosts->query()` из объекта произвольного запроса `$myPosts` в примере, чтобы выбрать содержимое, отображаемое в вашем произвольном цикле.

Отображает запись по ID записи:

```
$myPosts = new WP_Query( 'p=1' );
```

Отображает пять последних записей, пропуская первую:

```
$myPosts = new WP_Query( 'posts_per_page=5&offset=1' );
```

Отображает все записи за текущий день:

```
$today = getdate(); // получает текущую дату
$myPosts = new WP_Query('year=' .$today["year"]
    .'&monthnum=' .$today["mon"] .'&day=' .$today["mday"] );
```

Отображает все записи от 31 октября 2013 года:

```
$myPosts = new WP_Query( 'monthnum=10&day=31&year=2013' );
```

Отображает все записи категории ID 5 с меткой `bacon`:

```
$myPosts = new WP_Query( 'cat=5&tag=bacon' );
```

Отображает все записи с меткой bacon, за исключением записи категории с ID 5:

```
$myPosts = new WP_Query( 'cat=-5&tag=bacon' );
```

Отображает записи с метками writing или reading:

```
$myPosts = new WP_Query( 'tag=writing,reading' );
```

Отображает все записи с метками writing, reading и tv:

```
$myPosts = new WP_Query( 'tag=writing+reading+tv' );
```

Отображает все записи с произвольным полем color (цвет) со значением blue (синий):

```
$myPosts = new WP_Query( 'meta_key=color&meta_value=blue' );
```

Разбиение на страницы в цикле

Если вам требуется разбить содержимое, полученное в результате выполения цикла, на страницы (со ссылками для навигации), то вам нужно предпринять еще несколько шагов. Разбиение на страницы сделано только для работы с глобальной переменной $wp_query. То есть оно работает в цикле по умолчанию, и требуется некоторая ловкость рук, чтобы заставить его работать в произвольных циклах. Вам нужно заставить WordPress думать, что ваш произвольный запрос на самом деле – $wp_query, чтобы разбиение на страницы заработало.

```
<?php
$temp = $wp_query;
$wp_query= null;
$paged = ( get_query_var( 'paged' ) ) ? get_query_var( 'paged' ) : 1;
$wp_query = new WP_Query( 'posts_per_page=5&paged='.$paged );
while ( $wp_query->have_posts() ) : $wp_query->the_post();
?>
      <h2>
      <a href="<?php the_permalink(); ?>"><?php the_title(); ?></a>
      </h2>
      <?php the_excerpt(); ?>
<?php endwhile; ?>
```

Сначала нужно сохранить оригинальную переменную $wp_query во временную переменную $temp. Затем вы приписываете $wp_query значение ноль, чтобы полностью очистить ее. Это один из нескольких случаев, когда переписывание значения глобальной переменной в WordPress приемлемо. Теперь установите ваш новый объект WP_Query в переменную $wp_query и выполните его, вызвав функцию объекта query() для выбора записей для произвольного цикла. Обратите внимание на переменную $paged, добавленную в конец запроса. Она хранит текущую страницу, используя функцию get_query_var(), поэтому WordPress знает, как отображать навигационные ссылки. Теперь отобразим навигационные ссылки при разбиении на страницы:

```
<div class="navigation">
  <div class="alignleft"><?php previous_posts_link( '&laquo; Previous' ); ?></div>
```

```
  <div class="alignright"><?php next_posts_link( 'More &raquo;' ); ?></div>
</div>
```

Наконец, нужно вернуть `$wp_query` оригинальное значение:

```
<?php
$wp_query = null;
$wp_query = $temp;
?>
```

Теперь ваш произвольный цикл будет включать в себя правильное разбиение на страницы в зависимости от возвращаемого контента.

Использование query_posts( )

Цикл можно настроить практически как угодно, устанавливая соответствующие наборы параметров. Хотя объект `WP_Query` — механизм наиболее широкого действия для извлечения контента из базы данных WordPress, есть и другие методы нижнего уровня, с которыми вы еще познакомитесь.

Функция `query_posts()` используется для простого изменения контента, возвращаемого циклом по умолчанию. В частности, можно менять контент, возвращаемый в `$wp_query` после выполнения предустановленного запроса базы данных, и выполнять запрос повторно, используя `query_posts()`. Обратная сторона использования `query_posts()` таким образом — обнуление результатов предустановленного запроса, кэшированных ранее, что сказывается на производительности базы данных. Функция `query_posts()` должна быть помещена непосредственно над началом цикла:

```
query_posts( 'posts_per_page=5&paged='.$paged );
if ( have_posts() ) :
   while ( have_posts() ) : the_post();
      // код внутри цикла (теги шаблонов, html и пр.)
   endwhile;
endif;
```

В этом примере WordPress дается команда отображать только пять записей.

Явно вызванный `query_posts()` перезаписывает оригинальный контент в виде записей, извлеченных для цикла. Это значит, что любой контент, который вы ожидали в виде возврата, до использования `query_posts()` возвращен не будет. Например, если переданный WordPress URL соответствует странице категории http://example.com/category/zombie/, ни одна из записей в категории «зомби» не будет возвращена в списке записей после вызова `query_posts()`, если только она не попадает в пять последних записей. Вы явно перезаписываете параметры запроса, установленные парсигом URL и обработкой по умолчанию, когда передаете строку запроса `query_posts()`.

Чтобы не терять контент изначального цикла, вы можете сохранить параметры запроса после парсинга, используя глобальную переменную `$query_string`:

```
// инициализируем глобальную переменную query_string
global $query_string;

// изменяем порядок сортировки изначального цикла
query_posts( $query_string . "&orderby=title&order=ASC" );
```

В предыдущем примере все записи из категории «зомби» будут по-прежнему видны, но они будут упорядочены по алфавиту в порядке возрастания. Эта техника используется для изменения оригинального контента цикла без его потери.

Вы также можете упаковать все параметры `query_posts()` в массив, чтобы облегчить управление. Ниже приведен пример того, как возвращать только приклеенные записи WordPress, используя массив `$args` для хранения значений параметров:

```
$args = array(
    'posts_per_page' => 1,
    'post__in'  => get_option( 'sticky_posts' )
);
query_posts( $args );
```

Если не найдено ни одной приклеенной записи, будет возвращена последняя запись. Функция `query_posts()` используется только для изменения цикла главной страницы. Она не предназначена для создания дополнительных произвольных циклов. Если вы хотите внести небольшие изменения в предустановленный запрос, например добавить записи определенной категории или метку к каждой отображаемой странице, то `query_posts()` является самым быстрым способом сделать это. Однако процесс не всегда проходит без побочных эффектов или предостережений:

- `query_posts()` изменяет глобальную переменную `$wp_query` и имеет другие побочные эффекты. Ее не следует вызывать более одного раза и использовать внутри цикла. Пример показывает обращение к `query_posts()` перед началом обработки записи, когда дополнительные параметры добавлены к строке запроса, но цикл еще не начал проходить через список возвращенных записей. Неоднократный вызов `query_posts()` или же ее вызов внутри цикла может привести к некорректной работе основного цикла и отображению неожиданного контента.
- `query_posts()` сбрасывает глобальный объект `$wp_query`, и в таком случае это может сделать недействительными значения условных меток, таких как `is_page()` или `is_home()`. Полная реализация объекта `WP_Query` устанавливает все условные метки соответствующим образом. Например, вы можете обнаружить, что с добавлением новых записей к выбранному контенту предустановленный запрос, содержавший только одну запись, и, соответственно, `is_single()` более недействительны.
- Вызов `query_posts()` выполняет другой запрос базы данных, делающий недействительными все кэшированные результаты первого предустановленного

запроса. Вы как минимум удваиваете число выполненных запросов базы данных и увеличиваете нагрузку на нее с каждым запросом MySQL. С другой стороны, предустановленный запрос уже был запущен на тот момент, когда вы обратились к предустановленному циклу, так что у вас было бы мало шансов обойти его при построении полностью произвольного основного цикла.

Использование get_posts( )

Есть более простая в использовании альтернатива `query_posts()` — функция `get_posts()`, которая возвращает «сырые» данные записи. Вы увидите функцию `get_posts()`, используемую на административных страницах для генерирования списка страниц определенного типа или внутри плагина для захвата «сырых» данных для набора записей и их проверки на наличие конфигураций, таких как общие элементы, метки или внешние ссылки, с целью отбраковки контента после быстрой обработки. Она не предназначена для отображения видимого пользователем контента, поскольку отключает большую часть обработки запроса и фильтрации, которые осуществляются в рамках более общего подхода `WP_Query`.

Чего особенно не хватает `get_posts()`, так это способности задавать все глобальные данные, необходимые для создания тегов шаблона, отображающих данные текущей записи. Одной из основных проблем является то, что не все теги шаблонов доступны `get_posts()` по умолчанию. Чтобы это исправить, нужно вызвать функцию `setup_postdata()` для заполнения тегами шаблона, чтобы использовать их в произвольном цикле. Приведенный ниже пример показывает, как вернуть единичную случайную запись, используя `get_posts()`:

```
<?php
$randompost = get_posts( 'numberposts=1&orderby=rand' );
foreach( $randompost as $post ) :
    setup_postdata( $post );
?>
<h1><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h1>
<?php the_content(); ?>
<?php endforeach; ?>
```

Вы заметите другое существенное отличие при использовании `get_posts()`: значение возвращается в виде массива. Код цикла `foreach` используется для прогона цикла через значения массива. Данный пример возвращает только одну запись, но если возвращается более одной, цикл будет запущен для каждой. Затем вызывается функция `setup_postdata()` для передачи данных тегам шаблона.

Помните, что вы также можете установить параметр `get_posts()`, используя массив:

```
<?php
$args = array(
    'numberposts' => 1,
    'orderby'  => rand
);
$randompost = get_posts( $args );
```

Хотя вы можете увидеть, что в более старых кодах используются конструкции `get_posts()` или `query_posts()`, `WP_Query` является предпочтительным подходом и должен стать основой синтаксиса для произвольных циклов. Однако иногда вам может понадобиться быстрый и грубый доступ, предоставляемый `get_posts()`, чтобы сгенерировать дополнительный контекст или данные для дальнейшей настройки вашего цикла или плагина.

Сброс запроса

При настройке цикла по умолчанию или создании произвольных циклов хорошая практика — сбрасывать данные цикла после того, как вы закончите. WordPress поддерживает две различные функции для этих целей: `wp_reset_postdata()` и `wp_reset_query()`.

Первый метод сброса данных записи — `wp_reset_postdata()`. Эта функция, по сути, восстанавливает глобальную переменную `$post` для текущей записи в главном запросе. Это предпочтительный метод при использовании `WP_Query` для создания произвольных циклов.

Например, предположим, что у вас есть следующий произвольный цикл в файле темы `header.php`:

```
<?php
$myPosts = new WP_Query( 'posts_per_page=1&orderby=rand' );
// Цикл
while ( $myPosts->have_posts() ) : $myPosts->the_post();
  ?><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a><br /><?php
endwhile;
?>
```

Он отображает случайную запись в заголовке темы. Этот код также изменит основной объект запроса для других циклов на странице. Данные оригинального запроса не будут доступны, что может дать неожиданные результаты на основном цикле записей вашей темы.

Чтобы решить эту проблему, поместите вызов `wp_reset_postdata()` непосредственно после произвольного цикла:

```
$myPosts = new WP_Query( 'posts_per_page=1&orderby=rand' );
// Цикл
while ( $myPosts->have_posts() ) : $myPosts->the_post();
  ?><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a><br /><?php
endwhile;
// Восстанавливаем значение глобальной переменной $postwp_reset_postdata();
```

Вызов этой функции восстановит переменную `$post` для текущей записи в запросе. Это устранит любые странности в основном запросе для страницы, которую вы просматриваете.

Второй метод для сброса данных записи — это функция `wp_reset_query()`. Время от времени вы можете сталкиваться с проблемой использования условных тегов для

страницы после создания произвольного цикла. Условные теги позволяют запускать различный код на различных страницах в WordPress, например при использовании условного тега `is_home()` для определения, просматриваете ли вы домашнюю страницу блога. Как было написано в разделе «Использование query_posts( )», эта проблема вызывается потенциальным изменением данных на выходе запроса базы данных после установки условных тегов на основе оригинального набора значений. Чтобы решить проблему, нужно вызывать `wp_reset_query()`. Эта функция должным образом восстановит оригинальный запрос, включая условные теги, установленные ранее во время парсинга URL.

Рассмотрим следующий пример:

```
<?php query_posts( 'posts_per_page=5' ); ?>
<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
    <a href="<?php the_permalink(); ?>"><?php the_title(); ?></a><br />
<?php endwhile; endif; ?>
<?php
if( is_home() && !is_paged() ):
    wp_list_bookmarks( 'title_li=&categorize=0' );
endif;
?>
```

Выполнение этого кода возвращает последние пять записей вместе со ссылками, сохраненными в менеджере ссылок WordPress. Проблема, с которой вы столкнетесь, заключается в том, что условный тег `is_home()` не будет интерпретироваться корректно. Это означает, что ссылки будут отображаться на каждой странице, а не только на домашней. Чтобы решить эту проблему, необходимо вставить `wp_reset_query()` непосредственно после Loop:

```
<?php query_posts( 'posts_per_page=5' ); ?>
<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
    <a href="<?php the_permalink(); ?>"><?php the_title(); ?></a><br />
<?php endwhile; endif; ?>
<?php wp_reset_query(); ?>
<?php
if( is_home() && !is_paged() ):
    wp_list_bookmarks( 'title_li=&categorize=0' );
endif;
?>
```

Теперь, после того как у вас есть надлежащим образом восстановленная копия цикла объекта `WP_Query`, условный тег `is_home()` будет соблюдаться и ссылки будут отображаться только на домашней странице сайта. Хорошая практика — добавлять `wp_reset_query()` после использования `query_posts()` в цикле, чтобы гарантировать отсутствие проблем в дальнейшем. Функция `wp_reset_query()` обычно вызывает `wp_reset_postdata()`, но она также делает еще один шаг. Функция обычно разрушает предыдущий запрос, перед тем как сбрасывать его. Короче говоря, `wp_reset_query()` должна всегда использоваться после цикла, созданного с помощью `query_posts()`, а `wp_reset_postdata()` должна всегда использоваться после цикла, созданного с помощью `WP_Query` или `get_posts()`.

Больше чем один цикл

Цикл можно использовать в теме и плагинах многократно. Это облегчает отображение различных типов контента в различных местах сайта WordPress. Возможно, вы хотите отображать самые последние записи блога под каждой страницей на вашем сайте. Вы можете сделать это, создав более сложные циклы, многократно прогоняющие списки записей, или генерируя множественные массивы записей для прогона цикла.

Вложенные циклы

Вложенные циклы могут быть созданы внутри шаблонов темы посредством использования комбинации основного цикла и отдельных копий `WP_Query`. Например, вы можете создать вложенный цикл для отображения связанных записей на основе меток записи. Ниже приведен пример создания вложенного цикла внутри основного цикла для отображения связанных записей на основе меток:

```
<?php
if ( have_posts() ) :
    while ( have_posts() ) :
        the_post();
        // код внутри цикла (теги шаблонов, html и пр.)
        ?>
        <h1><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h1>
        <?php
        the_content();
        $tags = wp_get_post_terms( get_the_ID() );
        if ( $tags ) {
            echo 'Related Posts';
            $tagcount = count( $tags );
            for ( $i = 0; $i < $tagcount; $i++ ) {
                $tagIDs[$i] = $tags[$i]->term_id;
            }
            $args=array(
                'tag__in' => $tagIDs,
                'post__not_in' => array( $post->ID ),
                'posts_per_page' => 5,
                'ignore_sticky_posts' => 1
            );
            $relatedPosts = new WP_Query( $args );
            if( $relatedPosts->have_posts() ) {
                // цикл внутри связанных метками записей
                while ( $relatedPosts->have_posts() ) :
                    $relatedPosts->the_post(); ?>
                    <p><a href="<?php the_permalink(); ?>">
                        <?php the_title(); ?></a></p>
                    <?php
                endwhile;
            }
        }
    endwhile;
endif;
?>
```

Этот код будет отображать все ваши записи как обычно. Внутри основного цикла вы проверяете, есть ли другие записи, содержащие те же метки, что и основная. Если есть, будет отображаться пять последних записей, соответствующих критерию связанности. Если подходящих записей нет, раздел связанных записей не будет отображаться.

Многопроходные циклы

Функция `rewind_posts()` используется для сброса запроса записи и счетчика цикла, позволяя выполнять другой цикл с использованием того же самого контента, что и в первом цикле. Поместите вызов этой функции непосредственно после первого цикла. Вот пример обработки контента основного цикла дважды:

```
<?php while ( have_posts() ) : the_post(); ?>
  <!-- content. -->
<?php endwhile; ?>
<?php rewind_posts(); ?>
<?php while ( have_posts() ) : the_post(); ?>
  <!-- content -->
<?php endwhile; ?>
```

Сложные запросы

Вы также можете выполнять более сложные запросы внутри цикла. Теперь создадим цикл, который будет сравнивать значение индивидуального поля с использованием параметра `meta_compare`:

```
$args =  array(
    'posts_per_page' => '-1',
    'post_type' => 'product',
    'meta_key' => 'price',
    'meta_value' => '13',
    'meta_compare' => '<='
);
$myProducts = new WP_Query( $args );
// Цикл
while ( $myProducts->have_posts() ) : $myProducts->the_post();
    ?><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a><br /><?php
endwhile;
// Восстанавливаем значение глобальной переменной $post
wp_reset_postdata();
```

Как вы видите, параметр `meta_compare` используется для отображения всех товаров с метазначением для цены ниже или равным (`<=`) 13. Параметр `meta_compare` может принимать все виды операторов сравнения: `!=`, `>`, `>=`, `<`, `<=`, и по умолчанию `=`.

Для более сложных запросов метаданных используется параметр `meta_query`. Теперь вы можете расширить предыдущий пример. Вместо простого возврата записей о товарах с ценой ниже или равной 13 вы также можете возвращать товары синего цвета:

```
$args = array(
    'post_type' => 'product',
    'meta_query' => array(
        array(
            'key' => 'color',
            'value' => 'blue',
            'compare' => '='
        ),
        array(
            'key' => 'price',
            'value' => '13',
            'type' => 'numeric',
            'compare' => '<='
        )
    )
);
$myProducts = new WP_Query( $args );
// Цикл
while ( $myProducts->have_posts() ) : $myProducts->the_post();
    ?><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a><br /><?php
endwhile;
// Восстанавливаем значение глобальной переменной $postwp_reset_postdata();
```

Обратите внимание, что параметр `meta_query` принимает массив параметров. В данном примере первая единица в массиве — это массив, проверяющий, что товары синие. Второй параметр — это массив, проверяющий, что цена ниже или равна 13.

Создание цикла с использованием метапараметров запроса может быть невероятно полезным. Это важный инструмент для создания сложных сайтов с различными параметрами метаданных.

Глобальные переменные

Глобальная переменная — это переменная с определенным значением, доступ к которой можно получить откуда угодно из среды выполнения WordPress. Эти переменные хранят все типы информации о контенте цикла, авторе и пользователях и специфическую информацию о копии WordPress, такую как подключение к базе данных MySQL. Глобальные переменные могут использоваться только для возврата данных, что означает, что в них никогда не следует записывать данные напрямую. Перезаписывание значений глобальной переменной может вызвать неожиданные результаты в WordPress, поскольку значительная часть ядра и расширенной функциональности зависит от этих значений, установленных в одном контексте и остающихся единообразными на протяжении запроса, загрузки страницы или работы с отдельной записью. Присваивание значений глобальным переменным чаще всего вызывает неожиданные побочные эффекты, которые практически всегда не являются тем, чего хотел бы пользователь или автор блога. Однако здесь глобальные переменные обсуждаются, чтобы пролить больше света на управление данными записи, и вы можете увидеть отрывки кода, которые используют эти функции для обработки записи вне цикла.

Данные записи

Вы узнали, что ключевым первым шагом в цикле является вызов `the_post()`. Вызвав ее единожды, вы получите доступ ко всем данным в WordPress, ассоциированным с отображаемой записью. Эти данные хранятся в глобальной переменной `$post`. Переменная `$post` хранит данные последней записи, отображенной на странице. Так что если ваш цикл отображает десять записей, переменная `$post` будет хранить данные для десятой отображенной записи.

Приведенный ниже пример показывает, как вы можете обращаться к глобальной переменной `$post` и отображать все значения в массиве, используя функцию PHP `print_r()`.

```
<?php
global $post;
print_r( $post );  // выводит данные, сохраненные в переменной $post
?>
```

Предыдущий код выдаст массив значений для глобальной переменной `$post`. Запись блога WordPress по умолчанию будет выглядеть примерно следующим образом:

```
stdClass Object
(
    [ID] => 1
    [post_author] => 1
    [post_date] => 2012-06-09 19:05:19
    [post_date_gmt] => 2012-06-09 17:23:50
    [post_content] => Welcome to WordPress. This is your first post.
        Edit or delete it, then start blogging!
    [post_title] => Hello world!
    [post_excerpt] =>
    [post_status] => publish
    [comment_status] => open
    [ping_status] => open
    [post_password] =>
    [post_name] => hello-world
    [to_ping] =>
    [pinged] =>
    [post_modified] => 2012-06-09 19:04:12
    [post_modified_gmt] => 2012-06-09 19:04:12
    [post_content_filtered] =>
    [post_parent] => 0
    [guid] => http://localhost/Brad/?p=1
    [menu_order] => 0
    [post_type] => post
    [post_mime_type] =>
    [comment_count] => 1
    [ancestors] => Array
        (
        )
    [filter] => raw
)
```

Как вы можете видеть, глобальная переменная `$post` содержит все виды данных для записи. Вы также можете отобразить отдельные фрагменты данных из массива, такие как название записи и контент:

```
<?php
global $post;
echo $post->post_title;     // выводит заголовок записи
echo $post->post_content;   // выводит содержание записи
?>
```

Доступ к содержимому через глобальную переменную `$post` означает, что вы обращаетесь к нефильтрованному контенту. То есть все плагины, которые обычно изменяют контент на выходе, не действуют на глобальную переменную. Например, если у вас в запись был встроен сокращенный код `[gallery]` для отображения всех изображений, загруженных для этой записи, возврат контента записи даст `[gallery]` вместо реальной галереи изображений.

Помните, что в WordPress есть теги шаблона, которые можно вызывать где угодно для получения этих значений, и в большинстве случаев теги шаблона — более предпочтительный механизм для получения этой информации. Например, если вам нужно получить постоянную ссылку записи, можно использовать следующий метод:

```
<?php
global $post;
echo get_permalink( $post->ID );  // отображает постоянную ссылку записи
?>
```

Он будет рассмотрен более подробно в разделе «Работа вне цикла» далее в этой главе.

Данные автора

`$authordata` — это глобальная переменная, которая хранит информацию об авторе отображаемой записи. Вы можете использовать эту переменную для отображения имени автора:

```
<?php
global $authordata;
echo 'Author: ' .$authordata->display_name;
?>
```

Переменная `$authordata` создается, когда `setup_postdata()` вызывается во время вызова функции `the_post()` в Loop. Это значит, что глобальная переменная `$authordata` не будет создана до первого запуска Loop. Другая проблема этого метода в том, что глобальные значения не проходят через фильтры зацепок. Это означает: ни один плагин, установленный для расширения функциональности, не будет работать.

Предпочтительный метод получения информации об авторе метаданных, как и в случае получения данных записи, — использовать доступные теги шаблона WordPress. Например, для отображения имени автора используйте код:

```
<?php
echo 'Author: ' .get_the_author_meta( 'display_name' );
?>
```

Функции `get_the_author_meta()` и `the_author_meta()` могут возвращать метаданные, касающиеся автора контента. Если этот тег шаблона используется внутри цикла, нет необходимости передавать параметр ID пользователя, если вне — ID пользователя требуется для определения, какие метаданные об авторе возвращать.

Данные пользователя

Глобальная переменная `$current_user` хранит информацию о пользователе, авторизованном в данный момент. Это учетная запись, с помощью которой вы в данный момент авторизованы в WordPress. Следующий пример показывает, как отображать имя авторизованного пользователя:

```
<?php
global $current_user;
echo $current_user->display_name;
?>
```

Это полезная техника, если вы хотите отображать приветственное сообщение для пользователей. Не забывайте, что отображаемое имя является именем пользователя по умолчанию. Чтобы отобразить приветственное сообщение для любого авторизованного пользователя, используйте код:

```
<?php
global $current_user;
if ( $current_user->display_name ) {
    echo 'Welcome ' .$current_user->display_name;
}
?>
```

Данные среды

В WordPress также есть глобальные переменные, созданные для определения браузера. Ниже приведен пример того, как вы можете определить версию браузера пользователя в WordPress, используя глобальные переменные:

```
<?php
global $is_lynx, $is_gecko, $is_IE, $is_opera, $is_NS4,
$is_safari, $is_chrome, $is_iphone;
if ( $is_lynx ) {
    echo "You are using Lynx";
}elseif ( $is_gecko ) {
    echo "You are using Firefox";
```

```
}elseif ( $is_IE ) {
    echo "You are using Internet Explorer";
}elseif ( $is_opera ) {
    echo "You are using Opera";
}elseif ( $is_NS4 ) {
    echo "You are using Netscape";
}elseif ( $is_safari ) {
    echo "You are using Safari";
}elseif ( $is_chrome ) {
    echo "You are using Chrome";
}elseif ( $is_iphone ) {
    echo "You are using an iPhone";
}
?>
```

Это невероятно полезно при создании сайта, когда необходимо включить специфические для браузера задачи или функциональность. Как и всегда, лучше ориентироваться на веб-стандарты и пойти в сторону меньшего количества поддерживаемых браузеров, а при некоторых обстоятельствах это может быть весьма выгодным. Например, вы можете использовать переменную `$is_iphone` для загрузки отдельного списка стилей для пользователей с iPhone.

WordPress поддерживает другую глобальную переменную для определения того, заходит ли пользователь с мобильного устройства, например смартфона или планшета. Эта глобальная переменная называется `$is_mobile`. Вместо того чтобы вызывать эту глобальную переменную напрямую, можно воспользоваться функцией под названием `wp_is_mobile()`. Она определяет, использует ли пользователь мобильное устройство. Если да, то функция вернет `true`; если нет — `false`:

```
if ( wp_is_mobile() ) {
    echo "You are viewing this website on a mobile device";
}else{
    echo "You are not on a mobile device";
}
```

WordPress также запоминает, на каком типе веб-сервера содержится сайт, используя глобальные переменные `$is_IIS` и `$is_apache`. Например:

```
<?php
global $is_apache, $is_IIS;
if ( $is_apache ) {
    echo "web server is running Apache";
}elseif ( $is_IIS ) {
    echo "web server is running IIS";
}
?>
```

В зависимости от того, на каком веб-сервере размещен сайт, можно запрограммировать различные результаты. Как разработчику вам необходимо иметь в виду, что ваши плагины и темы будут запускаться на копиях WordPress на разных веб-серверах. Поэтому вам может понадобиться проверять, что именно использует пользователь для реализации специфических задач.

Глобальные переменные или теги шаблона?

В общих случаях теги шаблона следует использовать во всех возможных случаях. Есть несколько вариантов, при которых использование тега шаблона невозможно. Тогда для доступа к необходимой информации применяются глобальные переменные. Также глобальные переменные прекрасно подходят для возврата нефильтрованных данных, при котором значения будут обходить любой плагин, обычно используемый для преобразования контента, после чего вы получите оригинальные значения для работы. После того как код получил или обработал оригинальное значение, вы можете заставить фильтры плагина работать далее, используя следующий код:

```
<?php apply_filters( 'the_content', $post->post_content );?>
```

Хотя это и будет обсуждаться при разговоре о работе вне цикла, вы можете получить доступ к глобальным переменным внутри цикла, но не забывайте использовать их в режиме «только чтение», поскольку их изменение может привести к негативным побочным эффектам.

Работа вне цикла

Иногда бывает необходимо получить доступ к общей информации о записи или к управлению информацией об отображаемой записи вне Loop. WordPress обеспечивает некоторые функции управления настройками записей для тонкого контроля над их отображением.

Наряду с доступом к глобальным переменным есть набор функций WordPress для возврата общей информации, не специфической для отдельной или в данной момент отображаемой записи. Ниже приведен список функций, часто используемых при работе вне Loop:

- `wp_list_pages()` — отображение списка записей как ссылок;
- `wp_list_categories()` — отображение списка категорий как ссылок;
- `wp_list_bookmarks()` — отображение ссылок, сохраненных во вкладке Ссылки (Links);
- `wp_tag_cloud()` — отображение облака меток из всех меток;
- `get_permalink()` — возврат постоянной ссылки записи;
- `next_posts_link()` — ссылка на отображение предыдущих записей;
- `previous_posts_link()` — ссылка на отображение следующих записей.

Вы уже видели, как можно создавать навигационные ссылки, используя `next_posts_link()` и `previous_posts_link()` в примере произвольного цикла. Теперь изучим некоторые из этих функций, чтобы понять, как именно они работают.

Чтобы отобразить список страниц в WordPress, вы можете использовать функцию `wp_list_pages()`. Она возвращает страницы в виде списка, поэтому важно заключить вызов функции в теги `<ul>`:

```
<ul>
    <?php wp_list_pages( 'title_li=' ); ?>
</ul>
```

Предыдущий код будет генерировать список страниц WordPress со ссылками. Обратите внимание: вы не задаете значение параметру `title_li`, что оставляет название по умолчанию для ваших страниц. Функция будет генерировать список меню следующим образом:

```
<ul>
    <li class="page_item page-item-1">
        <a href="http://example.com/about/" title="About">About</a>
    </li>
    <li class="page_item page-item-2">
        <a href="http://example.com/order/" title="Order">Order</a>
    </li>
    <li class="page_item page-item-3">
        <a href="http://example.com/contact/" title="Contact">Contact</a>
    </li>
</ul>
```

Чтобы генерировать меню страниц, вы также можете использовать функцию `wp_page_menu()`. У этой функции списка страниц есть несколько преимуществ. Первое — это новый параметр `show_home`, позволяющей ссылке на главную страницу (Home) автоматически добавляться к списку страниц. У вас также нет необходимости удалять использование названий `title_li`, как в предыдущем коде. Эта функция также помещает меню в индивидуальные теги `<div>`, класс которых вы можете установить:

```
<?php wp_page_menu( 'show_home=1&menu_class=my-menu&sort_column=menu_order' ); ?>
```

Другая распространенная функция для генерирования ссылок — это `wp_list_categories()`. Она составляет ссылки категорий и подкатегорий. Рассмотрим следующий пример:

```
<ul>
  <?php wp_list_categories( 'title_li=&depth=4&orderby=name&exclude=8,16,34' ); ?>
</ul>
```

Этот код генерирует список категорий со ссылками. Как и раньше, вы не задаете значение названию, отличному от предустановленного. Вы также выставляете глубину на 4. Параметр глубины контролирует, сколько уровней в иерархии категорий должно быть включено в список. Категории будут упорядочены по имени. Вы также исключаете три категории (8, 16 и 34) на основе их IDs.

Функции `next_posts_link()` и `previous_posts_link()` обычно используются непосредственно после завершения цикла. Эти две функции генерируют предыдущую и следующую ссылки для просмотра большего количества записей на сайте.

Обратите внимание, что функция `next_posts_link()` на самом деле возвращает предыдущие записи. Причина в том, что WordPress обеспечивает отображение записей в обратном хронологическом порядке, поэтому следующая страница записей будет содержать записи, расположенные ранее на временной оси.

Теперь представьте, что вы хотите загрузить отдельную запись вне Loop. Чтобы сделать это, вы используете функцию `get_post()` для загрузки данных записи. Вот пример загрузки данных записи для записи с ID 1031:

```
<?php
$my_id = 1031;
$myPost = get_post( $my_id );
echo 'Post Title: ' .$myPost->post_title .'<br />';
echo 'Post Content: ' .$myPost->post_content .'<br />';
?>
```

У функции `get_post()` есть только один параметр: ID записи, которую вы хотите загрузить. Вы должны передать переменную, содержащую целое для ID. Передача литерального целого (например, 5) вызовет фатальную ошибку. Второй необязательный параметр — то, в каком виде вы хотите получить возвращаемые результаты: как объект, как ассоциативный массив или как числовой массив. По умолчанию возвращается объект. Для возвращения ассоциативного массива запустите код:

```
<?php
$my_id = 1031;
$myPost = get_post( $my_id, ARRAY_A );
echo 'Post Title: ' .$myPost['post_title'] .'<br />';
echo 'Post Content: ' .$myPost['post_content'] .'<br />';
?>
```

Неважно, как возвращаются результаты, вызов `get_post()` возвращает «сырой» контент из базы данных WordPress. Фильтры и обработка, обычно применяемые внутри цикла, не будут использоваться для возвращаемого контента. Решение — в использовании функции `setup_postdata()` вместе с `get_post()` для установки глобальных данных записи и меток шаблона для использования с записью:

```
<?php
$my_id = 1031;
$myPost=get_post( $my_id );
setup_postdata( $myPost );
the_title();
the_content();
?>
```

Функция `get_post()` использует внутренний кэш объекта WordPress. Это значит, что если загружаемая запись уже находится в кэше, не нужно будет запускать повторный запрос базы данных. Легко заметить, насколько полезной может быть эта функция в вопросах быстрой и эффективной загрузки отдельной записи вне цикла.

Некоторые функции, которые могут быть использованы внутри цикла, также работают и вне его. Например, вы можете использовать функцию `the_author_meta()` для возврата специфических метаданных об авторе:

```
The email address for user id 1 is <?php the_author_meta( 'user_email', 1 ); ?>
```

Помните, что, вызывая функцию `the_author_meta()` вне цикла, вы должны определить ID автора, для которого хотите загрузить метаданные. Если вы вызываете эту функцию внутри цикла, вам не нужно определять ID, поскольку будут загружены данные автора текущей записи.

В WordPress также есть специфические функции для возврата произвольных данных о записи вне цикла. Например, вы можете использовать функцию `get_the_title()`, чтобы вернуть название записи на основе ее ID:

```
<?php
echo 'Title: ' .get_the_title( 1031 );
?>
```

Вы также можете использовать функцию для возврата метаданных записи (произвольные поля) отдельной записи. Чтобы сделать это, используйте функцию `get_post_meta()`:

```
<?php
echo 'Color: ' .get_post_meta( 1031, 'color', true );
?>
```

Функция `get_post_meta()` принимает три параметра: ID (записи), key и single. ID — это ID записи, для которой вы хотите загрузить метаданные. Key — имя метазначения, которое вы хотите загрузить. Третье необязательное значение определяет, возвращать результаты как массив или как единичный результат. По умолчанию выставлено значение `false`, и возвращается массив. Как вы можете видеть, для возврата только одного цвета можно выставить значение `true`.

Резюме

Эта глава охватывает базовые механизмы выбора и отображения контента WordPress и содержит путеводитель по ядру WordPress, позволяющий определить местоположение кода для внедрения функций. Истинная сила WordPress — в его расширяемости за счет плагинов и тем. Модель данных WordPress вы изучите подробнее в главе 6, в которой показывается, как различные единицы данных, сохраненные для всего контента, данные о пользователях и метаданные соотносятся между собой. В главе 7 будут рассмотрены пользовательские типы записей, пользовательские таксономии и метаданные, показывающие различные типы контента, которые вы можете определять и использовать в WordPress. Все это затем будет использовано в качестве основы для создания полноценного плагина в главе 8. Наряду с плагинами другим основным путем расширения и индивидуальной настройки WordPress являются темы. Вы более подробно рассмотрите некоторые варианты циклов в работе с шаблонами и отображением контента в главе 9.

6 Управление данными

В этой главе:

- Понимание базы данных WordPress
- Изучение взаимосвязей таблиц базы данных
- Работа с классом базы данных WordPress
- Отладка произвольных запросов

Почти любой сайт в Интернете в наше время подключен к базе данных, которая хранит информацию о нем. WordPress не исключение, его поддерживает серверное приложение базы данных MySQL. Эта база данных хранит все данные о сайте, включая контент, пользователей, ссылки, метаданные, настройки и прочее. В этой главе рассматривается, как и какие данные хранятся и как работать с ними в WordPress, чтобы создавать интересные сайты.

Схема базы данных

Копия WordPress по умолчанию содержит 11 таблиц базы данных. WordPress гордится своим малым весом, основой которого является база данных. Структура базы данных спроектирована быть минимальной, однако она дает бесконечную гибкость при разработке и конструировании WordPress. Чтобы понять схему базы данных, полезно рассмотреть ее диаграмму.

Рисунок 6.1 демонстрирует структуру и таблицы базы данных WordPress, созданной во время стандартной установки. Не забывайте, что плагины и темы могут создавать индивидуальные таблицы. WordPress Multisite также создает дополнительные таблицы, поэтому в базе данных WordPress может быть больше таблиц, чем по умолчанию.

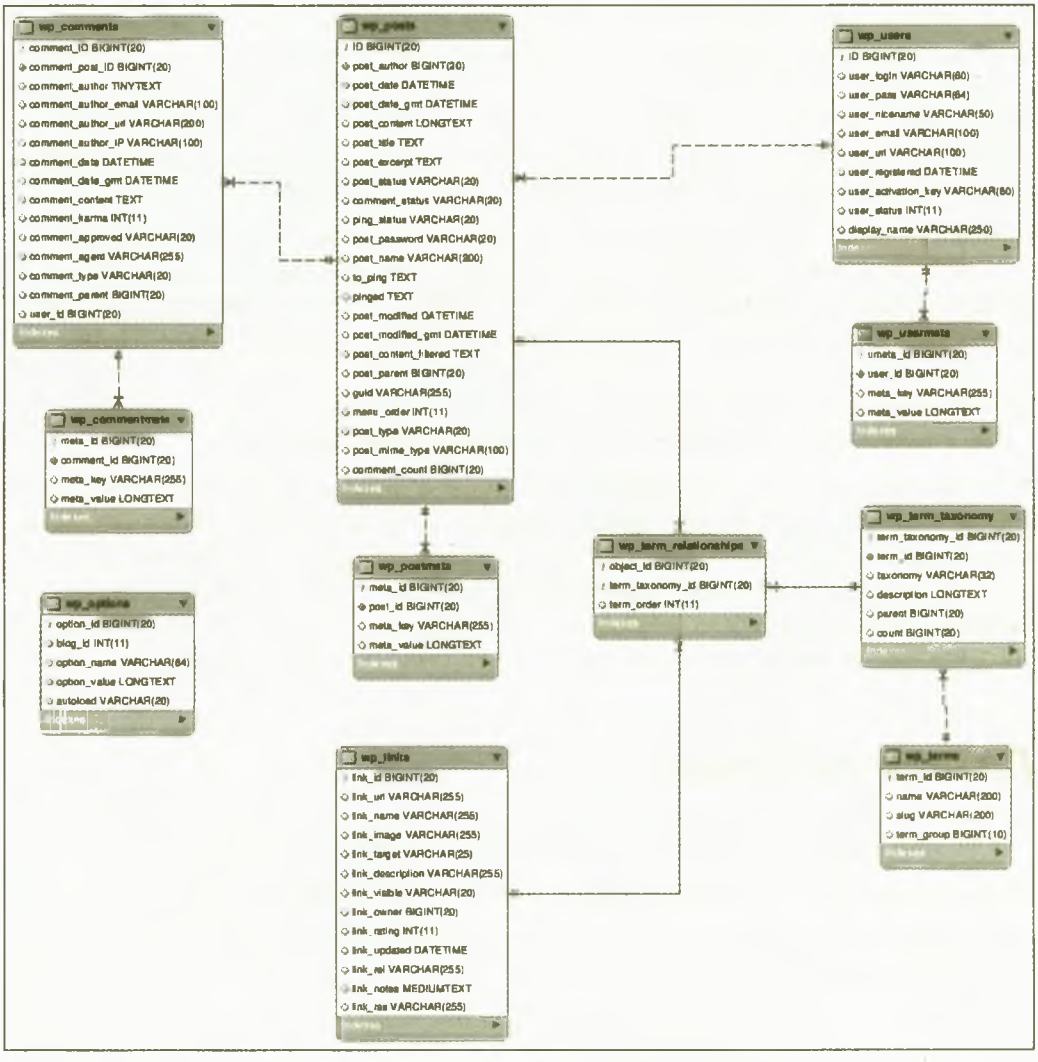


Рис. 6.1. Диаграмма базы данных WordPress

При выпуске новой версии WordPress обычно делается несколько изменений в базе данных. Чаще всего они незначительны. Например, смена типа данных поля таблицы или удаление поля, которое больше не используется. Обратная совместимость — основная зона внимания сообщества разработчиков WordPress, поэтому любые изменения, вносимые в базу данных, подробнейшим образом отслеживаются и редко затрагивают активные плагины или темы. Кодекс поддерживает крайне подробный журнал изменений, вносимых в базу данных, к которому вы можете обращаться при выпуске новой версии WordPress: http://codex.wordpress.org/Database_Description#Changelog.

Табличная структура WordPress весьма единообразна. Каждая таблица в базе данных содержит уникальное поле с ID, являющееся главным ключом к таблице, а также один или более индексов в полях, что улучшает скорость возврата данных при выполнении запросов. Как вы уже видели в главе 5, каждый круг цикла генерирует как минимум один, а возможно, и несколько запросов, направленных на извлечение записей, страниц и относящихся к ним метаданных или комментариев.

Самое важное поле в каждой таблице — поле, содержащее уникальный ID. Оно не всегда называется ID, но задается автоматически и используется для присвоения каждой записи в таблице уникального идентификатора. Например, при первой установке WordPress создается запись по умолчанию «Привет, мир!». Поскольку это первая запись, созданная в таблице `wp_posts`, ее ID — 1. Каждой записи присваивается уникальный ID, который может использоваться для загрузки информации о ней, а также как поле присоединения к другим таблицам в базе.

Есть только одно предупреждение, касающееся редакций, приложений и пользовательских типов записей. Любая единица этой информации сохраняется как новая запись в таблице `wp_posts` и получает уникальный ID. Это означает, что ID публикуемых записей могут отличаться друг от друга более чем на единицу. Например, у первой записи может быть ID 1, тогда как у второй — 15. Все зависит от того, как много дополнительной информации было добавлено между записями.

Детали таблицы

Сейчас для WordPress создано одиннадцать таблиц баз данных:

- `wp_commentmeta` — содержит все метаданные для комментариев.
- `wp_comments` — содержит все комментарии в WordPress, которые подключаются к записи по ее ID.
- `wp_links` — содержит все ссылки, добавленные через вкладку Ссылки.
- `wp_options` — хранит все параметры сайта, определенные во вкладке Параметры. а также параметры плагинов, тем и многое другое.
- `wp_postmeta` — содержит все метаданные записи (произвольные поля).
- `wp_posts` — содержит записи всех типов (предустановленных и избранных), страницы, медиафайлы и редакции. Чаще всего это самая большая таблица в базе данных.
- `wp_terms` — содержит все элементы таксономии, определенные для веб-сайта, соединяя их текстовые описания с идентификаторами, которые могут быть использованы как уникальные индексы в других таблицах.
- `wp_term_relationships` — соединяет элементы таксономии с контентом, создавая таблицу принадлежности. Отображает такие элементы, как название метки или категории, для соответствующей страницы или записи.

- `wp_term_taxonomy` — определяет таксономию, к которой приписан каждый элемент. Эта таблица позволяет иметь категории и метки с одинаковыми названиями, помещая их в разные таксономии.
- `wp_users` — содержит всех пользователей сайта (имя пользователя, пароль, электронный адрес).
- `wp_usermeta` — содержит метаданные пользователей (имя/фамилия, псевдоним, уровень пользователя и т. д.).

У каждой таблицы базы данных свое назначение в WordPress. В следующем разделе мы изучим несколько наиболее часто используемых таблиц и разберем примеры работы с ними.

Таблицы контента WordPress

Для получения основного контента сайта вы будете обращаться к таблице `wp_posts`. В ней хранятся все записи, страницы, приложения и редакции. Таблица содержит записи о приложениях, но не сами приложения. Физически они хранятся на сервере хостинга как стандартные файлы. Приведенный ниже запрос SQL является примером того, как извлекать все записи из базы данных, и представляет собой краткую версию того, что происходит при выполнении цикла по умолчанию в WordPress:

```
SELECT * FROM wp_posts
WHERE post_type = 'post'
AND post_status = 'publish'
ORDER BY post_date DESC
```

Этот запрос выбирает все записи в `wp_posts` с `post_type` — `'post'`. Поле `post_type` говорит о том, какой тип контента вы просматриваете. Чтобы вернуть все страницы, просто измените значение на `'page'`. В данном примере вы ищете только опубликованные записи, поэтому убедитесь, что `post_status` установлен на `'publish'`. Также вы сортируете записи таблицы в порядке убывания даты `post_date`, и записи будут отображаться в обратном хронологическом порядке. Данные запроса и имеющиеся вспомогательные инструменты мы обсудим далее.

Давайте изучим некоторые из наиболее полезных полей в таблице `wp_posts`. Вы уже знаете, что поле ID содержит уникальный ID записи. Поле `post_author` — это уникальный ID автора записи. Вы можете использовать его для возврата данных об авторе из таблицы `wp_users`. `post_date` — это дата создания записи. Поле `post_content` хранит основное содержимое записи или страницы, а `post_title` — название этого контента.

Крайне важным полем является `post_status`. На данный момент для записи WordPress определены семь статусов:

1. `publish` — опубликованная запись или страница.
2. `inherit` — редакция записи.
3. `pending` — запись, ждущая просмотра администратором или редактором.

4. `private` — личная запись.
5. `future` — запись, которая должна быть опубликована позже в соответствии с расписанием.
6. `draft` — запись в работе и является черновиком.
7. `trash` — контент в корзине и может быть восстановлен.

Статус записи имеет значение, когда при необходимости ограничить возможности автора записи публиковать или редактировать существующее содержимое используются роли. Роли подробнее обсуждаются в главе 12, а их воздействие на процесс управления контентом — в главе 14. Как и почти все в WordPress, пользовательские статусы записей могут быть созданы через плагины и темы.

`post_type` также хранится в этой таблице. Именно значение этой переменной определяет различные типы контента: записи, страницы, редакции и приложения. Начиная с версии WordPress 2.9 можно создавать индивидуальные типы записей, что дает бесконечное число возможностей при определении пользовательского контента в WordPress.

Таблица `wp_users` содержит данные по зарегистрированным учетным записям. Вновь мы видим поле ID, определяющее уникальный идентификатор информации о пользователе. `user_login` — это имя пользователя. Это значение пользователь должен ввести для авторизации в WordPress. Поле `user_pass` содержит phpass-пароль пользователя. Регистрационный электронный адрес пользователя находится в поле `user_email`. Поле `user_url` содержит сайт пользователя, а регистрационные данные хранятся в `user_registered`.

Теперь поговорим о таблице `wp_comments`. В ней хранятся все комментарии, пингбеки и трекбеки для веб-сайта.

Просматривая записи комментариев, вы увидите поле ID под названием `comment_ID`. Несмотря на это речь по-прежнему идет об уникальном идентификаторе записи в таблице. `comment_post_ID` — уникальный ID записи, к которой был добавлен комментарий. Помните, что по умолчанию вам не нужно авторизовываться, чтобы добавлять комментарии в WordPress. Поэтому вы увидите те же поля, что и в таблице пользователей.

В поле `comment_author` хранится имя комментатора. Если комментарий является пингбеком или трекбеком, в нем будет название записи, пославшей пинг. `comment_author_email` содержит электронный адрес комментатора, а его сайт хранится в `comment_author_url`. Другое важное поле — `comment_date`, в котором находится дата создания комментария. Оно используется для отображения комментариев в правильном порядке.

Таблицы таксономии WordPress

Элементы, взаимосвязи и таксономии распределены по трем таблицам, что позволяет создавать соотношения «много к одному» между категориями, метками, единицами

в индивидуальных таксономиях и записями. Эти взаимосвязи иерархические и многозначные. Хотя вы можете добавить массив идентификаторов меток или категорий, например, к каждой строке таблицы `wp_posts`, этот подход ограничивает число дескриптивных взаимосвязей для каждой записи, но отбирает лишнее пространство, выделяемое для меток и категорий, без чего можно было бы обойтись.

Если вы создаете категорию «копченый окорок» и помещаете в нее четыре записи, три таблицы с таксономией обновляются:

1. Одна строка таблицы `wp_terms` определяет «копченый окорок» и его слаг или сокращенную форму для URL. Это соотношение получает уникальный идентификатор (ключ) для соответствия элементам в других таблицах.
2. Одна строка таблицы `wp_term_taxonomy` передает «копченый окорок» и таксономию «категории». Это соотношение также получает уникальный ключ для комбинации «копченого окорока» и «категории». Если вы создаете еще и пользовательскую таксономию с записью «копченый окорок», в таблице `wp_term_taxonomy` будет еще одна строка для этого, также с уникальным ключом.
3. Четыре строки в таблице `wp_term_relationships` передают идентификатор «копченый окорок в категории» идентификаторам записи для каждой из записей в категории.

Рабочей лошадкой при работе с таблицами таксономии является оператор SQL `JOIN`, иногда называемый «продуктом» двух (или более) таблиц. `JOIN` создает временную таблицу, в которой каждая строка передается каждой строке во второй и последующих таблицах. После этого `WHERE`, часть операции `JOIN`, выбирает те строки, в которых совпадают те или иные поля. Чтобы найти все записи в категории «копченый окорок», WordPress сначала находит идентификатор пары «элемент–таксономия», выбирает соответствующие строки из таблицы `wp_term_relationships`, а затем применяет `JOIN` к `wp_posts` и выбранным из таблицы взаимоотношений строкам. Этот `JOIN` на языке SQL означает «извлечь все записи с идентификаторами в этот список», список формируется по ходу.

На рис. 6.2 графически представлено соединение таблицы `wp_posts` и таблиц таксономии.

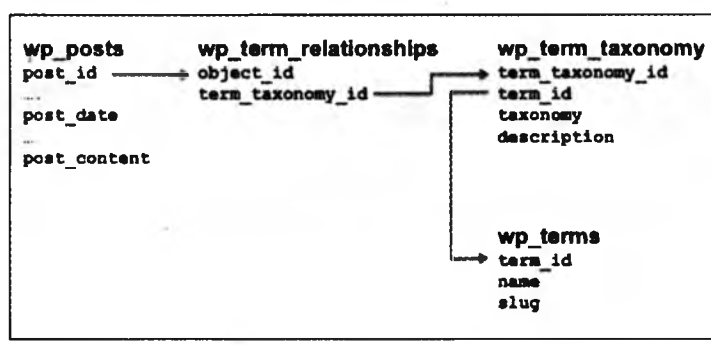


Рис. 6.2. Соотношение таблиц таксономии

SQL для выбора контента в связке с определенной меткой или категорией становится более сложным, поскольку мультитабличные операции `JOIN` для обеспечения соответствия «имени в таксономии во взаимосвязи» — это мощный инструмент, позволяющий снабжать контент богатыми многозначными описаниями, а также иметь независимые пространства названий категории, таксономии и метки.

Класс базы данных WordPress

В WordPress есть один класс объекта с функциями для работы непосредственно в базе данных. Этот класс базы данных называется `wpdb` и располагается в `wp-includes/wp-db.php`. Каждый раз, когда вы запрашиваете базу данных WordPress с помощью кода PHP, вы используете класс `wpdb`. Основная причина для использования этого класса — возможность выполнения WordPress ваших запросов максимально безопасным способом.

Простые запросы базы данных

При использовании класса `wpdb` использовать `$wpdb` можно будет только после ее определения как глобальной переменной. Чтобы сделать это, просто добавьте строку кода непосредственно перед любым вызовом функции `$wpdb`:

```
global $wpdb;
```

Одна из самых важных функций в классе `wpdb` — это функция `prepare()`. Она используется для исключения переменных, переданных запросам SQL. Это критически важный шаг в предотвращении атак типа SQL-инъекция на веб-сайт. Все запросы должны перед выполнением проходить через функцию `prepare`. Например:

```
<?php
$field_key = "address";
$field_value = "123 Elm St";
$wpdb->query( $wpdb->prepare( "INSERT INTO $wpdb->my_custom_table
    ( id, field_key, field_value ) VALUES ( %d, %s, %s )", 1,
      $field_key, $field_value ) );
?>
```

Этот пример добавляет данные в пользовательскую таблицу WordPress (отсутствующую в оригинальной копии WordPress), которая была создана предварительно. Используя `prepare()`, убедитесь, что вы заменили любые переменные в запросе на `%s` для строк и `%d` для целых. Затем перечислите переменные как параметры для функции `prepare()` в том же самом порядке. В примере `%d` представляет 1, `%s` представляет `$field_key`, а вторая `%s` представляет `$field_value`. Функция `prepare` применяется ко всем исходящим далее запросам.

Обратите внимание, что этот пример использует `$wpdb->my_custom_table` для обращения к таблице WordPress. Это имя переводится в `wp_my_custom_table`, если

wp_ — префикс таблицы. Это правильный способ определять корректный префикс таблицы при работе с таблицами в базе данных WordPress.

ЗАМЕЧАНИЕ

При установке WordPress вы можете задать индивидуальный префикс таблицы базы данных. По умолчанию это wp_, но многие предпочитают менять префикс по соображениям безопасности. Использование $wpdb-> — правильный способ определения префикса таблицы для любой копии WordPress.

Метод wpdb query() используется для выполнения простого запроса. Эта функция использует в первую очередь операторы SELECT и DELETE. Несмотря на название она предназначена не только для запросов SQL SELECT, но также выполняет любые предложения SQL для базы данных. Вот пример базовой функции запроса:

```
<?php
$wpdb->query( $wpdb->prepare( " DELETE FROM $wpdb->my_custom_table WHERE
id = '1' AND field_key = 'address' " ) );
?>
```

Как вы можете видеть, вы выполняете запрос с использованием функции query() класса wpdb для удаления field "address" с ID 1. Хотя функция query() позволяет выполнять любой запрос SQL базе данных WordPress, другие функции класса объекта базы данных лучше подходят для запросов SELECT. К примеру, функция get_var() используется для возвращения из базы данных единичной переменной:

```
<?php
global $wpdb;
$comment_count = $wpdb->get_var( $wpdb->prepare( "SELECT COUNT(*)
    FROM $wpdb->comments;" ) );
echo '<p>Total comments: ' . $comment_count . '</p>';
?>
```

В этом примере запрашивается подсчет всех комментариев в WordPress с отображением общего числа. Хотя возвращается только одна скалярная переменная, кэшируется весь блок результатов запроса. Лучше ограничить блок результатов, возвращаемый запросами, используя оператор WHERE, чтобы получать только те записи, которые вам нужны. В данном примере возвращаются все строки с записями о комментариях, даже если отображается их общее число. Это существенно влияет на использование памяти на больших сайтах.

Сложные операции с базой данных

Чтобы запросить целую строку таблицы, нужно использовать функцию get_row(). Функция get_row() может вернуть строку данных как объект, ассоциативный массив или численно индексированный массив. По умолчанию строка возвращается как объект, в данном случае как копия данных по записям. Пример:

```
<?php
$thepost = $wpdb->get_row( $wpdb->prepare( "SELECT *
```

```
    FROM $wpdb->posts WHERE ID = 1" ) );
echo $thepost->post_title;
?>
```

Запрашивается вся строка данных для записи с ID 1, отображается название записи. Свойства объекта `$thepost` — это имена колонок из запрошенной таблицы, в данном случае `wp_posts`. Чтобы вернуть результаты как массив, пошлите функции `get_row()` дополнительный параметр:

```
<?php
$thepost = $wpdb->get_row( $wpdb->prepare( "SELECT *
    FROM $wpdb->posts WHERE ID = 1" ), ARRAY_A );
print_r ( $thepost );
?>
```

Используя параметр `ARRAY_A` в `get_row()`, вы возвращаете данные в виде ассоциативного массива. В качестве альтернативы вы можете использовать параметр `ARRAY_N`, чтобы вернуть данные записи в виде численно индексированного массива.

Стандартные запросы `SELECT` должны использовать функцию `get_results()` для запроса множественных строк данных из базы данных. Следующая функция возвращает данные SQL как массив:

```
<?php
$liveposts = $wpdb->get_results( $wpdb->prepare( "SELECT ID, post_title
    FROM $wpdb->posts WHERE post_status = 'publish' " ) );
foreach ( $liveposts as $livepost ) {
    echo '<p>' .$livepost->post_title. '</p>';
}
?>
```

Предшествующий пример запрашивает все опубликованные записи WordPress и отображает названия записей. Результаты запроса возвращаются и хранятся как массив в `$liveposts`, который можно потом прогнать через цикл для отображения значений запроса.

Класс базы данных WordPress также поддерживает особые функции для операторов `UPDATE` и `INSERT`. Эти функции устраняют потребность в индивидуальных запросах SQL, поскольку WordPress будет сам создавать их на основе значений, переданных этим функциям. Вот структура функции `insert()`:

```
$wpdb->insert( $table, $data );
```

Переменная `$table` — это имя таблицы, в которую вы хотите внести значение. Переменная `$data` — это массив имен полей и данных для включения в эти имена полей. Например, если вы хотите включить данные в пользовательскую таблицу, сделайте следующее:

```
<?php
$newvalueone = 'Hello World!';
$newvaluetwo = 'This is my data';
$wpdb->insert( $wpdb->my_custom_table, array( 'field_one' => $newvalueone,
    'field_two' => $newvaluetwo ) );
?>
```

Первое, что вы делаете, — задаете две переменные для хранения данных, которые вы хотите добавить. После этого вы запускаете функцию `insert()`, передавая ей обе переменные в виде массива. Обратите внимание: `field_one` и `field_two` указываются в качестве двух полей, которые вы добавляете. Вы можете передать любое доступное поле таблицы, которую вы заполняете данными, чтобы данные поступали в это поле.

Функция `update()` работает крайне похоже на функцию `insert()`, за исключением того, что вам необходимо добавить выражение `WHERE`, чтобы WordPress знал, какие записи обновлять:

```
$wpdb->update( $table, $data, $where );
```

Переменная `$where` — это массив имен полей и данных для оператора SQL `WHERE`. Обычно он устанавливается на уникальный ID поля, которое вы обновляете, но также может содержать и другие имена полей таблицы.

```
<?php
$newtitle = 'My updated post title';
$newcontent = 'My new content';
$my_id = 1;
$wpdb->update( $wpdb->posts, array( 'post_title' => $newtitle,
    'post_content' => $newcontent ), array( 'ID' => $my_id ) );
?>
```

Сначала вы задаете обновленное название и переменные контента. Вы также задаете переменную `$my_id`, которая содержит ID обновляемой записи. Затем вы запускаете функцию `update()`. Обратите внимание, что третий посылаемый вами параметр — это массив, содержащий значения выражения `WHERE`, в данном случае ID записи. Предыдущий запрос обновляет заголовок и контент записи с ID 1. Помните, что при обновлении записи в таблице с помощью параметра `WHERE` вы можете установить множественные значения.

Показанные функции `insert()` и `update()` не должны оказываться внутри функции `prepare()`. Обе они, по сути, используют функцию `prepare()` после соединения запроса со значениями, переданными функции. Это гораздо проще, чем создавать запросы `INSERT` и `UPDATE` в WordPress вручную.

Работа с ошибками

В любой момент при работе с запросами приятно видеть сообщения об ошибках. По умолчанию при неудаче с пользовательскими запросами не возвращается ничего, поэтому сложно определить, что с ними не так. Класс `wpdb` предоставляет функции по отображению ошибок MySQL на странице. Вот примеры их использования:

```
<?php
$wpdb->show_errors();
$liveposts = $wpdb->get_results( $wpdb->prepare("SELECT ID, post_title
    FROM $wpdb->posts_FAKE WHERE post_status = 'publish'") );
$wpdb->print_error();
?>
```

Функцию `show_errors()` следует вызывать непосредственно перед выполнением запроса. Функцию `print_error()` следует вызывать непосредственно после выполнения запроса. При наличии ошибок в выражении SQL будут отображаться сообщения о них. Вы также можете вызвать функцию `$wpdb->hide_errors()`, чтобы скрыть все ошибки MySQL, или функцию `$wpdb->flush()`, чтобы удалить кэшированные результаты запроса.

Класс базы данных содержит дополнительные переменные, хранящие информацию о запросах WordPress. Вот список самых распространенных из них:

```
var_dump( $wpdb->num_queries ); // общее число запущенных запросов
var_dump( $wpdb->num_rows );    // общее число строк, возвращенных последним
                                // запросом
var_dump( $wpdb->last_result ); // результаты самого последнего запроса
var_dump( $wpdb->last_query );  // самый последний выполненный запрос
var_dump( $wpdb->col_info );    // информация столбца для самого последнего запроса
```

Вставьте приведенный выше код непосредственно после выполнения запроса, чтобы увидеть результаты. Это крайне полезно при определении причин неполадок в работе базы данных.

Еще одна мощная переменная базы данных — это `$queries`. Она хранит все запросы, запущенные WordPress. Чтобы активировать эту переменную, вы должны сначала установить значение постоянной `SAVEQUERIES` на `true` в файле `wp-config.php`. Это заставит WordPress сохранять все запросы, выполненные на каждой загруженной странице, в переменной `$queries`. Сначала добавьте эту строку кода в файл `wp-config.php`:

```
define( 'SAVEQUERIES', true );
```

Теперь все запросы будут сохраняться в переменной `$queries`. Вы можете отобразить всю информацию о них:

```
var_dump( $wpdb->queries ); // выводит все запросы,
                            // выполненные при загрузке страницы
```

Это особенно полезно, когда выявление неполадок замедляет время загрузки. Если плагин выполняет огромное число запросов, это может значительно замедлить время загрузки WordPress. Не забудьте отключить постоянную `SAVEQUERIES`, когда закончите просматривать запросы, поскольку хранение всех запросов также может замедлить время загрузки.

Как вы еще увидите, при разработке плагина или построении сложного цикла класс запроса базы данных — основной ресурс для работы с базой данных WordPress напрямую. Все ранее упомянутые функции класса базы данных используют специальные техники подготовки запросов, чтобы гарантировать, что все запросы выполняются наиболее безопасным образом. Как пошутил Рэндел Монро в своей xkcd-серии про «Little Bobby Tables» (xkcd #327), вряд ли вам захочется, чтобы пользователь смастерил объект для ввода, содержащий `DROP TABLES`, как умышленную SQL-инъекцию, что приведет к потере таблиц базы

данных WordPress. Функции подготовки запроса гарантируют, что вводимые данные не станут функциями SQL, вне зависимости от того, насколько мастерски они сделаны. Важно следовать этим методами запроса данных, чтобы быть уверенными, что ваш сайт самый эффективный и использует самые безопасные из возможных техник.

Прямое управление базой данных

Может случиться так, что вам захочется поработать с базой данных WordPress напрямую. Например, получить доступ к индивидуальным таблицам баз данных, созданным плагином или темой. Чтобы сделать это, необходимо использовать SQL для запроса данных из базы данных MySQL. Помните, что API WordPress обеспечивают доступ ко всем таблицам WordPress и только в исключительных случаях вам может понадобиться доступ к таблицам напрямую. Все примеры запросов в этой части используют для таблиц префикс `wp_`, но у таблиц вашей базы данных может быть другой префикс, определенный в файле `wp-config.php` при установке WordPress.

Один из наиболее распространенных методов работы с базой данных WordPress напрямую — это использование phpMyAdmin, как показано на рис. 6.3. Как было сказано в главе 3, phpMyAdmin — бесплатный инструментарий, предоставляемый большинством хостинговых компаний для администрирования баз данных MySQL через веб-интерфейс. Большинство примеров в этом разделе касаются прямого взаимодействия с MySQL, поэтому вам нужно использовать командную строку SQL для их выполнения. На рис. 6.3 показан стандартный вид базы данных при использовании phpMyAdmin.

Рис. 6.3. Просмотр базы данных WordPress через phpMyAdmin

Чтобы запустить выражения SQL в phpMyAdmin, просто выберите вкладку SQL в верхней части окна. Здесь вы можете выполнять любые запросы к базе данных WordPress. Мы всегда рекомендуем создавать запрос непосредственно

в phpMyAdmin, перед тем как передавать его скриптам PHP. Причина в том, что отладку выражений SQL гораздо быстрее проводить непосредственно в phpMyAdmin, чем при использовании кода PHP в WordPress. После того как вы довели запрос до совершенства, его можно использовать в коде PHP с уверенностью в том, что будут получены ожидаемые результаты. В приведенных ниже примерах вы будете запускать «сырые» запросы SQL. Помните, что если вы хотите запускать эти запросы в теме или плагине, вам нужно поместить их в класс базы данных WordPress.

Чаще всего требуется доступ к таблице `wp_posts`. Напоминаем: в ней хранятся все записи, страницы, индивидуальные типы записей, редакции и даже записи о приложениях. Различные типы контента определяются полем `post_type`. WordPress 2.9 предоставил разработчикам возможность определять пользовательские типы записей, что подробно обсуждается в главе 7. Это значит, что в поле могут быть дополнительные значения `post_type`. Чтобы увидеть все редакции записей в базе данных, выполните запрос:

```
SELECT * FROM wp_posts
WHERE post_type = 'revision'
```

Он возвращает все записи в `wp_posts`, для которых `post_type` определен как редакция. Вы можете изменить этот запрос, чтобы увидеть все приложения к записям, загруженные в WordPress:

```
SELECT guid, wp_posts.* FROM wp_posts
WHERE post_type = 'attachment'
```

В этом примере поле `guid` становится первым значением для возврата по запросу. Поле `guid` содержит полный URL файла приложения на сервере.

Таблица `wp_options` содержит все настройки, сохраненные для вашей копии WordPress. Параметры, сохраненные в этой таблице, сохраняются с `option_name` и `option_value`. Поэтому имя поля, которое вы вызываете, всегда будет представлять собой два этих имени, а не специфическое поле, основанное на значении параметра. Вот две крайне важные записи в этой таблице:

```
SELECT * FROM wp_options
WHERE option_name IN ( 'siteurl','home' )
```

Этот запрос возвращает две записи, одну со значением `option_name` в виде `home` и другую со значением `option_name` в виде `siteurl`. Это две настройки, сообщающие WordPress домен вашего сайта. Если вам нужно изменить домен, вы можете запустить запрос для обновления этих двух переменных:

```
UPDATE wp_options
SET option_value = 'http://yournewdomain.com'
WHERE option_name IN ('siteurl','home')
```

После запуска этого запроса ваш сайт немедленно стартует с новым доменом. Помните, что это просто обновление домена сайта в WordPress. URL приложений к записям и страницам также нужно обновить относительно нового домена.

Плагины тоже могут включать в себя информацию о домене, поэтому не забудьте провести проверку в среде разработки перед тем, как обновлять рабочий сайт. При запросе старого домена вы будете перенаправлены на новый. Если вы авторизованы, куки и сессия будут обнулены и вам придется авторизоваться заново. Это прекрасная техника, если вы создаете новый сайт на поддомене (например, http://new.example.com) и обновляете ссылки для передачи на рабочий веб-сайт.

В таблице `wp_options` есть и другие очень важные поля. Чтобы увидеть плагины, активные на сайте, посмотрите `active_plugins` `option_name`:

```
SELECT *
FROM wp_options
WHERE option_name = 'active_plugins'
```

Таблица параметров также хранит все параметры, определенные плагинами. У большинства активированных в WordPress плагинов есть что-то типа страницы настроек. Эти настройки обычно хранятся в `wp_options`, и плагины могут запрашивать их при необходимости. Например, плагин Akismet сохраняет параметр с названием `akismet_spam_count`, который содержит общее число спам-комментариев. Вы можете увидеть данный параметр с помощью следующего запроса:

```
SELECT * FROM wp_options
WHERE option_name = 'akismet_spam_count'
```

Таблица `wp_users` содержит информацию по всем пользователям, на данный момент существующим в WordPress. Если на вашем веб-сайте разрешена открытая регистрация, при присоединении новых пользователей они будут добавляться в эту таблицу. Таблица `wp_users` хранит крайне важную информацию о пользователях, включая имя пользователя, пароль, электронный адрес, URL сайта и дату регистрации. Скажем, вы хотите экспортировать все электронные адреса пользователей. Это легко можно сделать, запустив следующий запрос:

```
SELECT DISTINCT user_email
FROM wp_users
```

Теперь вы можете легко экспортировать все адреса электронной почты, загруженные в WordPress! Другим распространенным запросом, применяемым к `wp_users`, является восстановление пароля пользователя. Это можно сделать парой разных способов, но если вы совершенно отрезаны от WordPress, то можете восстановить пароль непосредственно в базе данных. Чтобы сделать это, необходимо обновить поле `user_pass` из командной строки MySQL:

```
UPDATE wp_users
SET user_pass = MD5('Hall0w33n')
WHERE user_login ='admin'
LIMIT 1;
```

Запуск этого запроса задает пароль администратора Hall0w33n. Обратите внимание, что новый пароль помещен в `MD5()`. Это конвертирует пароль в хэш MD5 hash. Начиная с версии WordPress 2.5 пароли содержат соль и хэшируются с использованием

библиотеки шифрования phpass, а не MD5. Не беспокойтесь, WordPress спроектирован так, что может определять хэшированные MD5 пароли и преобразовывать их в шифрование phpass. Так что показанный выше запрос успешно восстановит ваш пароль в WordPress.

В таблице `wp_comments` хранятся все комментарии к вашему сайту. Она содержит комментарии, авторов, URL сайтов, IP-адреса и многое другое. Вот пример запроса для отображения комментариев:

```
SELECT wc.* FROM wp_posts wp
INNER JOIN wp_comments wc ON wp.ID = wc.comment_post_ID
WHERE wp.ID = '1554'
```

Этот запрос возвращает все комментарии к записи с ID 1554. Другим важным полем в `wp_comments` является `user_id`. Если пользователь авторизован на вашем веб-сайте и добавляет комментарий, то это поле будет содержать его ID. Рассмотрим код, который отображает все комментарии, оставленные администратором:

```
SELECT wc.* FROM wp_comments wc
INNER JOIN wp_users wu ON wc.user_id = wu.ID
WHERE wu.user_login = 'admin'
```

В диаграмме базы данных на рис. 6.1 стрелками показаны соотношения между таблицами. Это невероятно полезно при написании индивидуальных запросов для возврата данных непосредственно из базы данных. Например, чтобы вернуть все комментарии для определенной записи, можно запустить запрос:

```
SELECT * FROM wp_comments
INNER JOIN wp_posts ON wp_comments.comment_post_id = wp_posts.ID
WHERE wp_posts.ID = '1'
```

Этот запрос возвращает все комментарии для записи с ID 1. Обратите внимание, как вы присоединяете поле `wp_comments.comment_post_ID` к полю `wp_posts.ID`. Необходим `JOIN` SQL, поскольку между комментариями и записями имеет место соотношение N:1. У каждой записи может быть много комментариев, но каждый комментарий может принадлежать только одной записи. Эти два поля показаны на диаграмме как соединенные поля таблицы. Рассмотрим также следующий пример, показывающий, как соединять таблицы `wp_users` и `wp_usermeta`:

```
SELECT * FROM wp_users
INNER JOIN wp_usermeta ON wp_users.ID = wp_usermeta.user_id
WHERE wp_users.ID = '1'
```

Как вы видите на диаграмме базы данных, поле `wp_users.ID` было соединено с полем `wp_usermeta.user_id`. Предыдущий запрос возвращает всю информацию о пользователе, включая метаданные, для пользователя с ID 1, по умолчанию являющегося учетной записью администратора. Еще раз: диаграмма базы данных позволяет невероятно легко определить, как таблицы соединяются по индексу значения внутри базы данных WordPress и как логические операции `INNER JOIN` могут построить наборы результатов связанных строк таблицы.

Если вы хотите узнать больше об SQL, здесь есть интересные руководства: http://www.w3schools.com/sql/default.asp.

Резюме

Эта глава охватывает схему базы данных WordPress, соотношение таблиц баз данных, класс базы данных WordPress и правильный способ отладки запросов базы данных. Вне зависимости от того, работаете вы с темами, плагинами или индивидуальными функциями, важно знать, как работать с базой данных WordPress. Понимание того, где и как WordPress хранит данные в базе данных, может помочь при разработке более сложных возможностей сайта.

Далее вы изучите пользовательский контент в WordPress, используя пользовательские типы записей. Вы также рассмотрите пользовательские таксономии, метаданные и поймете, почему и то и другое так важно при разработке сайтов на WordPress.

Пользовательские типы записей, пользовательские таксономии и метаданные

7

В этой главе:

- Понимание и создание пользовательских типов записей
- Отображение и использование контента записей пользовательского типа
- Создание и использование пользовательских таксономий
- Понимание и использование метаданных

Самая важная часть любого веб-сайта на WordPress — контент. В WordPress по умолчанию определены различные типы контента и таксономий, но нередко вам требуется определить собственные типы контента для построения именно того сайта, который вам нужен.

За последние несколько лет WordPress представил несколько весьма продвинутых и простых в использовании инструментов для работы со всеми типами произвольного контента. Это помогло ему развиться в полноценную систему управления контентом, способную поддерживать абсолютно любые типы сайтов, вне зависимости от содержимого.

В этой части вы узнаете, как создавать пользовательские типы записей и контента в WordPress. Вы также изучите работу с пользовательскими таксономиями для группировки и классификации контента. Наконец, вы выясните, как присоединять произвольные фрагменты метаданных к контенту.

Понимание данных в WordPress

При работе с различными типами данных в WordPress важно понимать, что это за данные и как они могут быть индивидуализированы. В базовой копии WordPress есть пять предустановленных типов записей:

1. **Post (Запись).** Записи или статьи в блоге, обычно упорядоченные по дате.
2. **Page (Страница).** Иерархические статичные страницы с контентом.
3. **Attachment (Приложение).** Медиафайлы, загруженные в WordPress и прикрепленные к записям, такие как изображения и файлы.
4. **Revision (Редакция).** Редакции записей, используемые как резервные копии, которые могут быть восстановлены при необходимости.
5. **Nav Menus (Меню навигации).** Пункты меню, добавленные в меню навигации с использованием функции управления меню WordPress.

Для базового блога или небольшого сайта достаточно этих предустановленных типов записей. Однако если вы планируете построение более сложного сайта с системой управления контентом CMS, вам могут понадобиться пользовательские типы записей.

Что такое пользовательский тип записи?

Пользовательский тип записи в WordPress — это фрагмент контента, заданный пользователем. Эти типы записей позволяют определить любой тип контента в WordPress, чтобы не ограничиваться только предустановленными типами записей, перечисленными в предыдущем разделе. Это дает бесконечное число возможностей.

Среди возможных идей для индивидуальных типов записей:

- продукты;
- события;
- видео;
- благодарности;
- цитаты;
- журнал ошибок.

Не забывайте, что пользовательские типы записей могут быть абсолютно чем угодно, не только доступными публично фрагментами контента. Например, вы можете установить пользовательский тип записи как журнал ошибок, чтобы отслеживать ошибки в приложении. Здесь вы ограничены только собственным воображением.

Регистрация пользовательского типа записей

Для создания нового пользовательского типа записи вы будете использовать функцию `register_post_type()`:

```
<?php register_post_type( $post_type, $args ); ?>
```

Функция `register_post_type()` принимает два параметра:

1. `$post_type` — название типа записи. Может содержать только строчные буквы, пробелы запрещены, максимальная длина — 20 символов.
2. `$args` — массив аргументов, которые определяют тип записи и различные параметры в WordPress.

Теперь посмотрим на базовый пример регистрации пользовательского типа записи. Вы можете зарегистрировать тип записи в WordPress в двух разных местах: первое — это файл темы `functions.php`; второе — это пользовательский плагин. Вы можете добавить приведенный ниже код и в пользовательский плагин, но для данного примера внесем его в файл темы `functions.php`.

```
<?php
add_action( 'init', 'prowp_register_my_post_types' );
function prowp_register_my_post_types() {
    register_post_type( 'products',
        array(
            'labels' => array(
                'name' => 'Products'
            ),
            'public' => true,
        )
    );
}
?>
```

Теперь зайдите в консоль WordPress. Вы увидите, что в ней ниже вкладки **Комментарии (Comments)** появилась вкладка **Products (Продукты)**, как показано на рис. 7.1. Это новый пользовательский тип записи, который вы зарегистрировали с помощью предыдущего кода.

Как видите, WordPress автоматически создает пользовательский интерфейс администратора для нового пользовательского типа записи. Новый пункт меню позволяет вам создавать новые записи типа «Продукты», а также редактировать существующие записи наравне с прочими в WordPress. Это базовый пример, но вы уже видите, насколько легко задавать произвольный контент в WordPress.

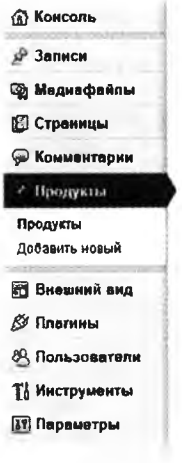


Рис. 7.1. Пользовательский тип записи «Продукты»

ЗАМЕЧАНИЕ

При регистрации пользовательских типов записей нужно всегда использовать зацепку `init`. Это первая зацепка, доступная после полной инициализации WordPress, которая будет проверять, что пользовательский тип записи достаточно рано регистрируется в процессе.

При регистрации пользовательского типа записи доступно множество аргументов. Для того чтобы ими пользоваться, важно их понимать.

public

Определяет, доступен тип записи публично в консоли администратора или в пользовательском интерфейсе сайта. По умолчанию выставлено значение `false`, скрывающее тип записи от просмотра. Предустановленные значения для `show_ui`, `exclude_from_search`, `publicly_queryable` и `show_in_nav_menus` зависят от этой настройки.

show_ui

Определяет, создается ли по умолчанию пользовательский интерфейс в консоли администратора WordPress для управления этим типом записи. По умолчанию значение определяется аргументом `public`.

publicly_queryable

Определяет, может ли контент записи этого типа быть публично запрошен через пользовательский интерфейс сайта. Если выставлено значение `false`, все запросы от пользователей будут возвращать ошибку 404, поскольку запрос запрещен. По умолчанию значение определяется аргументом `public`.

exclude_from_search

Позволяет исключить записи пользовательского типа из результатов поиска WordPress. По умолчанию значение определяется аргументом `public`.

show_in_nav_menus

Определяет, будет ли тип записи доступен для выбора при управлении меню WordPress. По умолчанию значение определяется аргументом `public`.

supports

Этот аргумент позволяет определить, какие метаполя появляются на экране при создании или редактировании записи нового типа (по умолчанию `title` и `editor`):

- `title` — задает название записи.
- `editor` — отображает редактор контента на экране редактирования записи с загрузчиком медиафайлов.
- `author` — выбирает поле для определения автора записи.
- `thumbnail` — задает метаполе миниатюры для записи.
- `excerpt` — отображает редактор цитат на экране редактирования типа записи.

- `comments` — определяет, будут ли активированы комментарии для записи этого типа.
- `trackbacks` — определяет, будут ли активированы трэкбеки и пингбеки.
- `custom-fields` — отображает метаполе зоны редактирования произвольных полей.
- `page-attributes` — отображает поле атрибутов для выбора порядка записи. Для этого аргументу `hierarchical` следует задать значение `true`.
- `revisions` — отображает поле редакций записи.
- `post-formats` — отображает метаполе форматов записи с зарегистрированными форматами записи.

labels

Определяет массив ярлыков, описывающих тип записи в консоли администратора. См. раздел «Определение ярлыков типа записи» далее в этой главе.

hierarchical

Аргумент `hierarchical` позволяет задать, является ли тип записи иерархическим, подобно страницам WordPress. `hierarchical` позволяет иметь древовидную структуру контента записей данного типа. По умолчанию аргументу присвоено значение `false`.

has_archive

Позволяет типу записи иметь архивную страницу. Архивная страница типа записи подобна странице записей WordPress, отображающей последние записи в блоге. Это позволяет отображать список записей данного типа в порядке, определенном в файле шаблона темы.

can_export

Определяет, будет ли контент записей данного типа доступен для экспорта с использованием встроенной возможности экспорта WordPress (Инструменты ▸ Экспорт). По умолчанию аргументу присвоено значение `true`.

taxonomies

Дает название массиву зарегистрированных таксономий для прикрепления к индивидуальному типу записей. Например, вы можете передать в этом массиве `category` и `post_tag`, чтобы прикрепить предустановленные таксономии рубрик и меток к вашему типу записи. По умолчанию к пользовательскому типу записи никакие таксономии не прикреплены.

menu_position

Позволяет выбрать позицию, в которой пользовательский тип записи будет отображаться в меню администратора. По умолчанию новые типы записей отображаются после вкладки Комментарии.

menu_icon

Устанавливает индивидуальную иконку в меню для вашего типа записи. По умолчанию используется иконка для записей.

show_in_menu

Определяет, будет ли отображаться меню администратора для вашего типа записи. Аргумент принимает три значения: `true`, `false` или строку. Строка может быть либо страницей верхнего уровня, как `tools.php`, либо `edit.php?post_type=page`. Вы также можете задать строке параметр `menu_slug`, чтобы добавить пользовательский тип записи как объект подменю в существующем пользовательском меню. По умолчанию определяется значением аргумента `show_ui`.

show_in_admin_bar

Определяет, будет ли отображаться пользовательский тип записи в панели администратора WordPress. По умолчанию определяется значением аргумента `show_in_menu`.

capability_type

Дает название строке или массиву характеристик для этого типа записи. По умолчанию приписывается значение `post`.

capabilities

Это массив индивидуальных характеристик, необходимых для редактирования, удаления, просмотра и публикации записей данного типа.

query_var

Этот аргумент устанавливает переменную запроса записей данного типа. По умолчанию его значение `true` и устанавливается для `$post_type`.

rewrite

Аргумент `rewrite` создает уникальные постоянные ссылки для этого типа записей. Это позволит вам сделать индивидуальным слаг типа записи в URL. Аргументу могут быть присвоены значения `true`, `false` или массив значений.

- `slug` — определяет индивидуальный слаг постоянной ссылки. Значение по умолчанию для `$post_type`.
- `with_front` — определяет, будет ли тип записи использовать основу постоянных ссылок. Например, если префиксом ваших постоянных ссылок является `/blog`, а значение `with_front` определено как `true`, постоянные ссылки записей этого типа будут начинаться с `/blog`.
- `pages` — определяет, будет ли постоянная ссылка обеспечивать разбиение на страницы. Значение по умолчанию `true`.
- `feeds` — определяет, будет ли постоянная ссылка на фид встроена в записи этого типа. Значение по умолчанию совпадает со значением `has_archive`.

По умолчанию значение аргумента определено как `true`, а `$post_type` используется в качестве слага.

Этот раздел охватывает множество аргументов типа записи. Ниже приведены наиболее распространенные примеры их использования.

```
<?php
add_action( 'init', 'prowp_register_my_post_types' );
function prowp_register_my_post_types() {
  $args = array(
        'public' => true,
        'has_archive' => true,
        'taxonomies' => array( 'category' ),
        'rewrite' => array( 'slug' => 'product' ),
        'supports' => array( 'title', 'editor', 'author', 'thumbnail', 'comments' )
    );
    register_post_type( 'products', $args );
}
?>
```

В этом примере вы сначала определяете, что тип записи будет публичным. Вы также указываете, что у типа записи есть архивная страница, приписывая аргументу `has_archive` значение `true`. Используя аргумент `taxonomies`, вы прикрепляете предустановленную таксономию Category к индивидуальному типу записи Products.

В данном примере вы хотите изменить слаг постоянной ссылки для вашего типа записи. Вместо http://example.com/products/zombie-bait, используя предустановленный слаг `products` из имени типа записи, вы задаете слаг типа записи для отдельного `product`. Будет сгенерирована постоянная ссылка типа http://example.com/product/zombie-bait. Это делается с использованием аргумента `rewrite` и определением индивидуального слага для вашего типа записи. В итоге вы устанавливаете аргумент `supports`. Этот код добавляет название, редактора, автора, миниатюру и метаполе комментариев к экрану создания и редактирования записи пользовательского типа.

Для получения более подробной информации по функции `register_post_type()` обратитесь к официальной странице Кодекса http://codex.wordpress.org/Function_Reference/register_post_type.

ЗАМЕЧАНИЕ

При регистрации нового пользовательского типа записи важно сбросить правила преобразования в WordPress. Вы можете сделать это, вызвав функцию `flush_rewrite_rules()` в модуле активации плагина или же вручную, через Параметры ▸ Постоянные ссылки и сохранение настроек постоянных ссылок. Это поможет избежать ошибок 404 по постоянным ссылкам записей нового типа.

Определение ярлыков типа записи

При создании пользовательского типа записи в WordPress для него показывается несколько строк в консоли администратора. Эти строки обычно представляют собой ссылку, кнопку или дополнительную информацию о типе записи. По умолчанию термин «запись» используется для неиерархического типа записей, а термин «страница» — для иерархического.

Например, если вы используете приведенный ранее в этой части базовый код регистрации пользовательского типа записи, то при добавлении нового товара в верхней части страницы увидите текст «Add New Post» («Добавить новую запись»). Причина в том, что товар — это запись типа Product. Это не совсем точно, потому что вы на самом деле добавляете не запись, а новый товар. Настройка аргумента `labels` (ярлыки) при регистрации пользовательского типа записи позволит вам точно определять, как этот тип отображается на экране и в меню.

Среди доступных ярлыков для пользовательского типа записи:

- `name` — общее название типа записи, обычно во множественном числе. Используется в консоли WordPress, а также другими плагинами и темами.
- `singular_name` — название в единственном числе. Используется в консоли WordPress, а также другими плагинами и темами.
- `add_new` — ярлык для добавления объекта подменю «Add New» («Добавить новую»). Текст по умолчанию: «Add New».
- `add_new_item` — заголовок на главной странице со списком записей для добавления новой записи. Текст по умолчанию: «Add New Post/Page» («Добавить новую запись/страницу»).
- `edit_item` — используется для редактирования отдельной записи. Текст по умолчанию: «Edit Post/Page» («Редактировать запись/страницу»).
- `new_item` — текст для создания новой записи. Текст по умолчанию: «New Post/Page» («Новая запись/страница»).
- `view_item` — текст для отображения отдельной записи. Текст по умолчанию: «View Post/Page» («Просмотр записи/страницы»).
- `search_items` — текст, отображаемый для поиска записей этого типа. Текст по умолчанию: «Search Posts/Pages» («Поиск записей/страниц»).

- `not_found` — текст, отображаемый, если ни одна запись не найдена. Текст по умолчанию: «No posts/pages found» («Записей/страниц не найдено»).
- `not_found_in_trash` — текст, отображаемый, если в корзине не найдено ни одной записи. Текст по умолчанию: «No posts/pages found in Trash» («Записей/страниц в корзине не найдено»).
- `parent_item_colon` — текст, отображаемый при показе родительской страницы. Используется только для записей иерархического типа, текст по умолчанию «Parent Page» («Родительская страница»).

Настройка каждого значения делает администрирование WordPress более приятным. В следующем коде производится изменение оригинального кода регистрации пользовательского типа записи и настраиваются ярлыки для типа записи Product:

```
<?php
add_action( 'init', 'prowp_register_my_post_types' );
function prowp_register_my_post_types() {
    $labels = array(
        'name' => 'Products',
        'singular_name' => 'Product',
        'add_new' => 'Add New Product',
        'add_new_item' => 'Add New Product',
        'edit_item' => 'Edit Product',
        'new_item' => 'New Product',
        'all_items' => 'All Products',
        'view_item' => 'View Product',
        'search_items' => 'Search Products',
        'not_found' =>  'No products found',
        'not_found_in_trash' => 'No products found in Trash',
        'parent_item_colon' => '',
        'menu_name' => 'Products'
    );
    $args = array(
        'labels'    =>    $labels,
        'public'    =>    true
    );
    register_post_type( 'products', $args );
}
?>
```

Работа с пользовательскими типами записи

Теперь, когда вы понимаете, как регистрировать пользовательский тип записи, давайте изучим, как использовать его при разработке сайта на WordPress. Обычно это задача темы — отображать записи на титульной странице сайта. Однако это не всегда так, поскольку некоторые из пользовательских типов записей не требуют публичного отображения — например, журнал ошибок. Все зависит от того, какова функция этого типа записи.

Чтобы отобразить данные записи пользовательского типа, можно использовать произвольный цикл `WP_Query`, приведенный как пример в главе 5. Помните, что

WP_Query принимает параметр post_type, который определяет, какой тип контента возвращать. В приведенном далее примере возвращаются все записи типа Product в WordPress:

```
$args =  array(
    'posts_per_page' => '-1',
    'post_type' => 'products',
);
$myProducts = new WP_Query( $args );
// Цикл
while ( $myProducts->have_posts() ) : $myProducts->the_post();
    ?><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a><br /><?php
endwhile;
// Восстанавливаем значение глобальной переменной $post
wp_reset_postdata();
```

Обратите внимание, что параметр post_type определен как products, что является значением параметра $post_type при регистрации пользовательского типа записи Products.

Теперь изменим произвольный цикл, чтобы возвращать только товары рубрики Specials:

```
$args =  array(
    'posts_per_page' => '-1',
    'post_type' => 'products',
    'tax_query' => array(
        array(
            'taxonomy' => 'category',
            'field' => 'slug',
            'terms' => 'specials'
        )
    )
);
$myProducts = new WP_Query( $args );
```

Используя параметр tax_query в WP_Query, произвольный цикл вернет только записи типа Product, относящиеся к рубрике Specials.

Вы можете использовать все методы создания произвольных циклов с WP_Query, которые описаны в главе 5, чтобы отображать записи пользовательского типа. Насколько полезны пользовательские записи в WordPress, становится понятно при разработке более сложных сайтов.

Файлы шаблона записи пользовательского типа

Ранее в этой части при регистрации пользовательского типа записи вы изучили has_archive. Активация этого аргумента позволит вам создавать файл по шаблону архива, который будет отображать все записи пользовательского типа по умолчанию. Шаблон архива для записи пользовательского типа должен быть назван по стандарту archive-{post-type}.php. Например, шаблон архива для

пользовательского типа записи Product следует назвать `archive-products.php`. Этот шаблон архива — прекрасное место для отображения всех товаров.

Как и в случае с шаблоном архива, WordPress также распознает единичный шаблон для записей вашего типа. Это шаблон, загружаемый при просмотре отдельной записи пользовательского типа. Шаблон записи пользовательского типа должен быть назван по стандарту `single-{posttype}.php`. Таким образом, шаблон записи пользовательского типа Product следует назвать `single-products.php`. При просмотре отдельного товара по ссылке, такой как http://example.com/products/zombie-bait, будет загружен шаблон `single-products.php`.

Файлы шаблона темы, включая пользовательские типы записей, будут детально рассмотрены в главе 9.

Особые функции типа записи

WordPress поддерживает множество разных особых функций типа записи, что существенно облегчает работу. В данном разделе мы рассмотрим некоторые из наиболее широко используемых функций, которые пригодятся вам при создании сайтов.

Чтобы вернуть список всех зарегистрированных типов записей в WordPress, используйте функцию `get_post_types()`.

```
<?php get_post_types( $args, $output, $operator ); ?>
```

Эта функция принимает три необязательных параметра:

1. `$args` — массив аргументов, соответствующих типу записи.
2. `$output` — тип возвращаемых данных: `names` или `objects`. По умолчанию `names`.
3. `$operator` — оператор для использования множественных `$args`. По умолчанию `and`.

Дл возврата списка записей пользовательского типа, зарегистрированных в WordPress, используйте функцию `get_post_types()` следующим образом:

```
$args = array(
    'public'   => true,
    '_builtin' => false
);
$post_types = get_post_types( $args, 'names', 'and' );
foreach ( $post_types  as $post_type ) {
    echo '<p>'. $post_type. '</p>';
}
```

Как показано в коде выше, вы используете два аргумента в массиве `$args`: `public` и `_builtin`. Аргумент `public` задает возврат только тех записей пользовательского типа, которые отображаются публично. Аргументу `_builtin` присвоено значение `false`, то есть предустановленные типы записей, такие как записи и страницы, возвращаться не будут. Вы также настраиваете аргумент `$output` на возврат только

имени записи и аргумент `$operator` на использование «and» («и») для множественных `$args`, передаваемых функции.

Чтобы определить, записью какого типа является фрагмент контента, используйте функцию `get_post_type()`:

```
<?php get_post_type( $post ); ?>
```

Она принимает только один параметр — `$post` — объект записи или ID записи.

Вы можете отобразить тип записи, используя следующий код:

```
<?php echo 'The post type is: '.get_post_type( $post->ID ); ?>
```

Возможно, вам захочется поработать с индивидуальным типом записей, созданным плагином или темой. В первую очередь всегда следует убедиться, что этот тип существует. Для этого используйте функцию `post_type_exists()`.

```
<?php post_type_exists( $post_type ); ?>
```

Функция принимает только один обязательный параметр — `$post_type` — тип записей, наличие которого вы хотите проверить.

Если вы хотите проверить, существует ли пользовательский тип записей `products`, используйте код:

```
if( post_type_exists( 'products' ) ) {
  echo 'The Products post type exists';
}
```

Еще одна полезная функция при работе с пользовательскими типами записей — `add_post_type_support()`. Она позволяет регистрировать некоторые возможности для типа записей, такие как метаполе для миниатюры.

```
<?php add_post_type_support( $post_type, $supports ) ?>
```

Это полезная функция на тот случай, если существующий тип записи не поддерживает нужную вам возможность. Функция `add_post_type_support()` принимает два параметра:

1. `$post_type` — название типа записей, с которым вы работаете.
2. `$supports` — строка или массив добавляемой возможности.

В качестве примера предположим, что тип записей `products` не поддерживает миниатюру или комментарии. Чтобы добавить поддержку обеих возможностей, используем код:

```
add_post_type_support( 'products', array( 'thumbnail', 'comments' ) );
```

Эта функция крайне полезна, если вам необходимо работать с пользовательским типом записей, определенным в отдельном плагине или теме. Вместо того чтобы взламывать регистрационный код в плагине или теме, можно использовать

функцию `add_post_type_support()` и активировать любую возможность, необходимую для работы кода.

WordPress также поддерживает функцию изменения типа записи. Это `set_post_type()`.

```
<?php set_post_type( $post_id, $post_type ); ?>
```

Функция принимает два параметра:

1. `$post_id` — ID записи, которую вы хотите обновить (обязательное поле).
2. `$post_type` — имя типа записей, на который надо сменить текущий (необязательное поле со значением по умолчанию `post`).

Таксономия WordPress

Таксономия определяется как способ группировки схожих объектов. В первую очередь таксономии дают возможность определять взаимосвязи внутри контента сайта. В случае WordPress для группировки записей вы используете рубрики и метки. Делая это, вы определяете таксономию записей. Таксономия может быть иерархической (рубрики и подрубрики), но это не обязательно, как в случае с метками.

Предустановленные таксономии

По умолчанию WordPress предоставляет три таксономии:

1. **Рубрика (Category)** — сегмент для группировки схожих записей.
2. **Метка (Tag)** — метка, прикрепляемая к записи.
3. **Рубрика ссылок (Link category)** — сегмент для группировки схожих ссылок.

Рубрики являются иерархическими и определяются при создании записи. *Метки* не используют иерархию и также определяются при создании записи. *Рубрики ссылок* используются при группировке схожих ссылок через менеджер ссылок WordPress. Три предустановленные таксономии доступны для использования в копии WordPress с настройками по умолчанию.

Каждая создаваемая рубрика или метка является элементом этой таксономии. Например, рубрика «Музыка» является элементом таксономии рубрик, метка «Кетчуп» — элемент таксономии меток. Понимание таксономии и ее элементов поможет вам при определении пользовательских таксономий в WordPress.

Понимание того, как можно классифицировать контент с использованием единой структуры таксономий, существенно облегчит структуризацию контента сайта на WordPress с самого начала. Разработка рамок единой таксономии дает легкий и прозрачный доступ к информации на веб-сайте.

Структура таблиц таксономии

WordPress содержит три таблицы базы данных, в которых хранится информация по таксономии: `wp_terms`, `wp_term_relationships` и `wp_term_taxonomy`. Эта схема таксономии, добавленная в WordPress 2.3, делает функциональность таксономии в WordPress невероятно гибкой, то есть вы можете создавать и определять любые типы пользовательской таксономии для использования на сайте.

В таблице `wp_terms` хранятся все элементы таксономии. Это могут быть рубрики, метки, рубрики ссылок и определенные вами пользовательские элементы таксономии. Таблица `wp_term_taxonomy` определяет, к какой таксономии какой элемент относится. Например, в этой таблице будут перечислены все ID меток со значением таксономии `post_tag`. Если вы создали пользовательскую таксономию, ее значение станет ее именем. Таблица `wp_term_relationships` представляет собой таблицу соотношений и объединяет элементы таксономии с контентом. Например, при присвоении записи метки здесь создается новая запись, объединяющая ID записи и ID элемента.

Понимание соотношений в таксономии

Чтобы в полной мере понять соотношения между таблицами таксономии, полезно посмотреть на диаграмму структуры таблиц таксономии в базе данных на рис. 7.2.

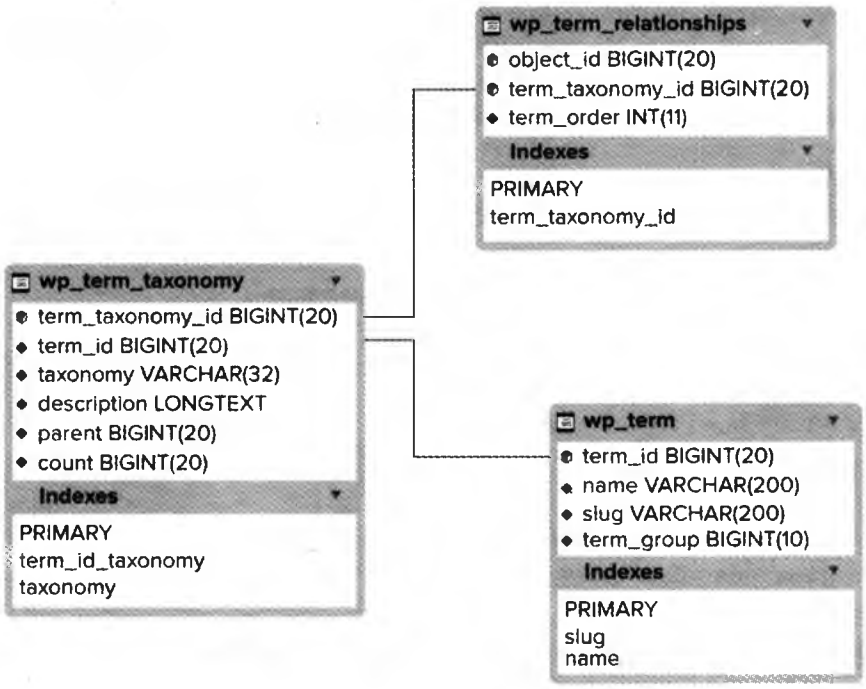


Рис. 7.2. Структура таблиц таксономии

Как вы видите, три таблицы таксономии соединяются уникальными ID. Ниже приведен запрос для отображения всех записей вместе с прикрепленными к ним элементами таксономии:

```
SELECT wt.name, wp.post_title, wp.post_date FROM wp_terms wt
INNER JOIN wp_term_taxonomy wtt ON wt.term_id  = wtt.term_id
INNER JOIN wp_term_relationships wtr ON wtt.
   term_taxonomy_id = wtr.term_taxonomy_id
INNER JOIN wp_posts wp ON wtr.object_id = wp.ID
WHERE wp.post_type = 'post'
```

Обратите внимание, как вы соединяете поля таблицы, что отображено на рис. 7.2. Предыдущий пример возвращает только три поля: элемент таксономии, название записи и данные записи. Этот же запрос возвращает все записи из базы данных WordPress, а также прикрепленные к ним элементы таксономии.

ЗАМЕЧАНИЕ

Чтобы узнать больше о соотношениях таблиц таксономии и выяснить, почему WordPress необходимо разделять эти многозначные отношения по разным таблицам, обратитесь к разделу о таблицах таксономии WordPress в главе 6 «Управление данными».

Построение собственных таксономий

У создания собственных таксономий множество преимуществ. Представьте, что у вас есть блог-сайт о еде. При создании новых записей вы хотите пометить некий рецепт как азиатский, но также маркировать его по ингредиентам, температуре и времени приготовления и т. д. Создание пользовательских таксономий дает вам свободу определения этих различных методов систематизации контента и расширяет WordPress от программного обеспечения для ведения блогов до полноценной системы управления контентом (CMS).

Обзор пользовательских таксономий

После исправления схемы таксономии в WordPress 2.3 у вас есть возможность определять пользовательские таксономии для контекста. WordPress сделал создание и интеграцию новых таксономий простым как никогда.

В WordPress 2.8 была добавлена возможность автоматического отображения метаполя на экран редактирования типа записи для добавления элементов таксономии непосредственно в записи. WordPress также создает иконку в меню для доступа к новой панели управления таксономиями для администрирования элементов таксономии.

Создание индивидуальных таксономий

Пришло время построить первую пользовательскую таксономию! Вы собираетесь создать первую таксономию для пользовательского типа записи `product`,

который вы зарегистрировали ранее. Если вы продаете товары онлайн, вам необходимо найти способ группировать те или иные типы товаров. Вы собираетесь сделать пользовательскую таксономию, чтобы определять каждый тип товара в WordPress.

Сначала вам нужно определить новую таксономию с использованием функции WordPress `register_taxonomy()`. Эта функция позволит вам определить, как новая таксономия будет выглядеть и работать. Приведенный ниже код будет работать и в индивидуальном плагине, но мы используем его в находящемся в папке темы файле `functions.php`. Откройте `functions.php` и добавьте в него следующий код:

```
<?php
add_action( 'init', 'prowp_define_product_type_taxonomy' );
function prowp_define_product_type_taxonomy() {
    register_taxonomy( 'type', 'products', array( 'hierarchical' => true,
        'label' => 'Type', 'query_var' => true, 'rewrite' => true ) );
}
?>
```

Определение таксономии начинается с обработчика `init`, который предписывает WordPress выполнить функцию `prowp_define_product_type_taxonomy()` во время инициализации. Затем эта функция вызывает функцию WordPress `register_taxonomy()`. Она используется для создания пользовательской таксономии на основе переданных вами значений.

Теперь следует разобрать параметры, передаваемые функции `register_taxonomy()`. Первый параметр — это название таксономии, в данном случае `type`. Это название, определяющее таксономию в базе данных. Второй параметр — тип объекта. В данном примере вы будете использовать `products`, название вашего пользовательского типа записей. Третий и последний параметр — для аргументов, передаваемых в виде массива.

В данном примере вы передаете четыре аргумента. Первый — `hierarchical`, определяющий, будет или нет новая таксономия поддерживать вложенные таксономии, формируя иерархию. В предыдущем примере значение было установлено на `true`, поэтому таксономия будет функционировать подобно встроенным рубрикам WordPress, содержащим много подрубрик. Следующий аргумент — `label` — используется для определения названия таксономии на страницах администратора WordPress. Если аргументу `query_var` присвоено значение `false`, то таксономия не будет принимать никакие запросы, если `true` — то имя (с тире вместо пробелов) таксономии используется как запрашиваемая переменная в строках URL. Определение значения строки для `query_var` перезаписывает предустановленный вариант. Например, `query_var => 'strength'` позволит строкам URL формы `example.com/?strength=weapons` использоваться для выбора контента из индивидуальной таксономии.

Последний аргумент — `rewrite`, которому присваивается значение `true`. Это сообщает WordPress, следует ли ему использовать красивые постоянные ссылки при просмотре пользовательской таксономии. Присваивая ему значение `true`, вы

можете получить доступ к записям с пользовательской таксономией по ссылке типа example.com/type/weapons, а не уродливой example.com/?type=weapons.

Теперь, когда вы создали пользовательскую таксономию для type, давайте посмотрим, что с ней сделал WordPress. Первое, что вы заметите в консоли администратора, — это новая ссылка для таксономии «Тип» («Type») в пункте меню «Продукты» («Products»), как показано на рис. 7.3.

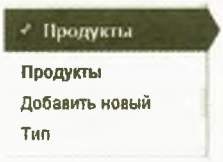


Рис. 7.3. Параметры меню для пользовательской таксономии

Переход по этой ссылке в меню приводит вас на панель администратора для пользовательской таксономии типов, показанную на рис. 7.4. Здесь вы можете создавать новые элементы таксономии, редактировать и удалять существующие элементы, выяснять, как много товаров приписано к каждому из них, а также искать элементы таксономии.

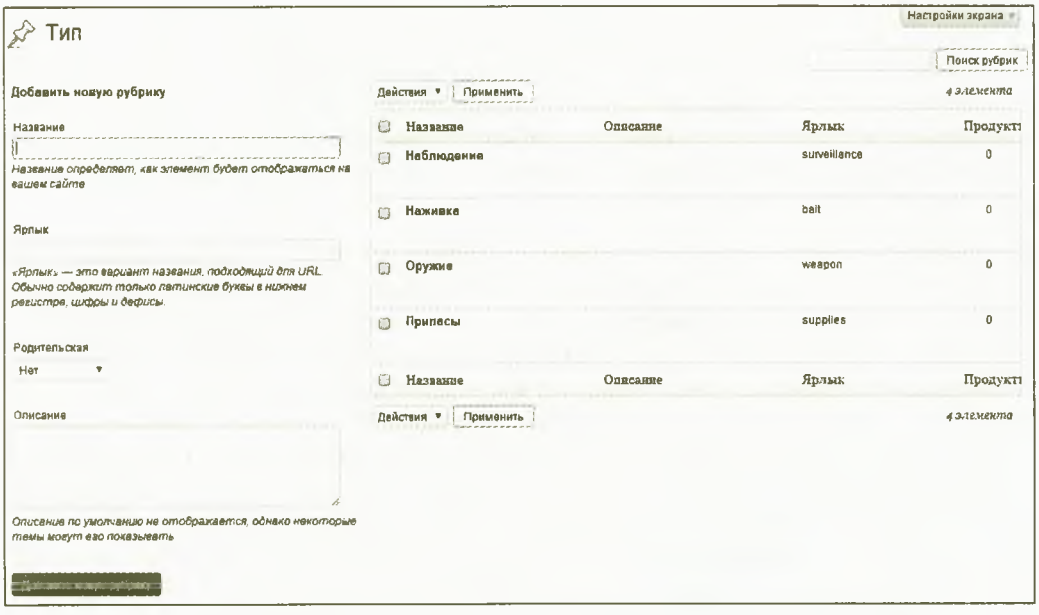


Рис. 7.4. Панель администратора для пользовательской таксономии

Последним новым элементом, добавляемым для пользовательской таксономии, является метаполе на экране редактирования товара, показанное на рис. 7.5. Чтобы увидеть его, пройдите по ссылке «Добавить новый продукт» («Add New Product»).

Метаполе появляется с правой стороны вашего экрана и выглядит похожим на метаполе для рубрикации. Здесь вы можете легко добавлять и удалять новые типы товаров.

Как и в случае с пользовательскими типами записей, при регистрации пользовательской таксономии можно задать множество различных аргументов:

- `public` — определяет, доступна ли таксономия публично в консоли администратора или в пользовательском интерфейсе сайта. По умолчанию выставлено значение `true`. Предустановленные значения для `show_ui` и `show_in_nav_menus` зависят от этой настройки.
- `show_ui` — определяет, создается ли по умолчанию пользовательский интерфейс в консоли администратора WordPress для управления этой таксономией. По умолчанию значение определяется аргументом `public`.

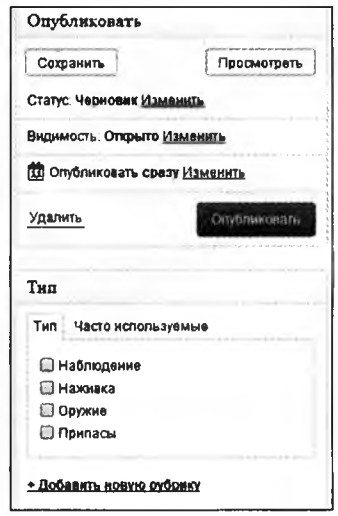


Рис. 7.5. Метаполе пользовательской таксономии

- `show_in_nav_menus` — определяет, будет ли таксономия доступна для выбора при управлении из меню WordPress. По умолчанию значение определяется аргументом `public`.
- `show_tagcloud` — определяет, может ли встроенный виджет «Облако меток» («Tag Cloud») использовать эту таксономию. По умолчанию значение определяется аргументом `show_ui`.
- `hierarchical` — определяет, будет ли пользовательская таксономия иерархической (как рубрики) или неиерархической (как метки). По умолчанию аргументу присвоено значение `false`.
- `update_count_callback` — функция, которая вызывается, когда происходит обновление счетчика элемента таксономии. Предустановленное значение: none.

- `query_var` — активирует публичный запрос `var` для таксономии. Может принимать значения `true`, `false` или строку для установки индивидуального значения запроса `var`.
- `rewrite` — аргумент `rewrite` создает уникальные постоянные ссылки для этой таксономии. Это позволит вам сделать индивидуальным слаг таксономии в URL. Аргументу могут быть присвоены значения `true`, `false` или массив значений. При передаче массива принимаются следующие значения:
  - `slug` — определяет индивидуальный слаг постоянной ссылки. Предустанавливает значение `name` таксономии.
  - `with_front` — определяет, будет ли таксономия использовать основу постоянных ссылок. Например, если префиксом ваших постоянных ссылок является `/blog`, а значение `with_front` определено как `true`, постоянные ссылки таксономии будут начинаться с `/blog`.
  - `hierarchical` — позволяет иерархические URL. Значение по умолчанию: `false`.

По умолчанию значение аргумента определено как `true`, а `$taxonomy` используется в качестве слага.

Для получения более подробной информации по функции `register_taxonomy()` обратитесь к официальной странице Кодекса http://codex.wordpress.org/Function_Reference/register_taxonomy.

Определение ярлыков пользовательской таксономии

Так же как и в случае создания пользовательского типа записей в WordPress, пользовательские таксономии поддерживают несколько текстовых строк, которые отображаются в консоли администратора. Эти строки обычно представляют собой ссылку, кнопку или дополнительную информацию о пользовательской таксономии. По умолчанию понятие *метка* (Tag) используется для неиерархических таксономий, а понятие *рубрика* (Category) — для иерархических таксономий.

Среди доступных ярлыков для пользовательской таксономии:

- `name` — общее название таксономии, обычно во множественном числе.
- `singular_name` — название в единственном числе.
- `search_items` — текст для кнопки поиска.
- `popular_items` — ярлык для популярных объектов.
- `all_items` — ярлык для всех объектов.
- `parent_item` — текст для родительского объекта. Не используется для неиерархических таксономий.
- `parent_item_colon` — то же, что и `parent_item`, но с двоеточием в конце.
- `edit_item` — используется для редактуры отдельного объекта таксономии.

- `update_item` — используется как текст для обновления объекта пользовательской таксономии.
- `add_new_item` — текст для создания нового объекта таксономии.
- `new_item_name` — текст для присвоения названия новому объекту.
- `separate_items_with_commas` — текст о разделении объектов запятыми, используемый в метаполе. Неприменим к иерархическим таксономиям.
- `add_or_remove_items` — текст, отображаемый в метаполе таксономии при деактивации JavaScript. Неприменим к иерархическим таксономиям.
- `choose_from_most_used` — выборка из текстов, наиболее часто используемых в метаполе таксономии. Неприменим к иерархическим таксономиям.
- `menu_name` — текст пунктов меню. По умолчанию имеет значение `name`.

Присвоение этих меток существенно облегчает работу по администрированию элементов пользовательской таксономии. Теперь изменим регистрационный код использовавшейся ранее пользовательской таксономии с помощью пользовательских ярлыков:

```
<?php
add_action( 'init', 'prowp_define_product_type_taxonomy' );
function prowp_define_product_type_taxonomy() {
    $labels = array(
        'name' => 'Type',
        'singular_name' => 'Types',
        'search_items' => 'Search Types',
        'all_items' => 'All Types',
        'parent_item' => 'Parent Type',
        'parent_item_colon' => 'Parent Type:',
        'edit_item' => 'Edit Type',
        'update_item' => 'Update Type',
        'add_new_item' => 'Add New Type',
        'new_item_name' => 'New Type Name',
        'menu_name' => 'Type'
    );
    $args = array(
        'labels' => $labels,
        'hierarchical' => true,
        'query_var' => true,
        'rewrite' => true
    );
    register_taxonomy( 'type', 'products', $args );
}
?>
```

Использование пользовательской таксономии

Теперь, когда вы создали пользовательскую таксономию, необходимо понять, как использовать ее на сайте. Как и всегда, WordPress предоставляет некоторые простые в использовании функции для работы с пользовательской таксономией.

Ниже вы можете увидеть, как отобразить облако меток, показывающее элементы пользовательской таксономии:

```
<?php wp_tag_cloud( array( 'taxonomy' => 'type', 'number' => 5 ) ); ?>
```

Функция `wp_tag_cloud()` может принимать множество разных аргументов, но в данном примере вы используете всего два: `taxonomy` и `number`. Сначала вы задаете `type` таксономии, что заставляет WordPress возвращать только элементы таксономии, определенные в соответствии с созданной вами пользовательской таксономией для типов. После этого вы определяете число элементов, которое хотите отобразить, — в данном примере 5. Вызов этой функции на боковой панели темы отображает облако меток, показывающее пять наиболее популярных элементов таксономии (к которым приписано большинство товаров).

Вы также можете создать произвольный цикл, используя `WP_Query` для отображения товаров для определенного элемента таксономии. Скажем, вы хотите создать произвольный цикл для отображения товаров только типа `weapon`:

```
<?php
$args = array(
    'post_type' => 'products',
    'tax_query' => array(
    array(
        'taxonomy' => 'type',
        'field' => 'slug',
        'terms' => 'weapon'
        )
    )
);
$products = new WP_Query( $args );
while ( $products->have_posts() ) : $products->the_post();
    echo '<p>' .get_the_title(). '</p>';
endwhile;
wp_reset_postdata();
?>
```

Вот оно! Два аргумента `WP_Query`, посланные вами, — это `post_type`, в данном случае `products`, и `tax_query`, определяющий, какой элемент таксономии использовать.

Вы также без проблем можете отобразить элементы пользовательской таксономии, присвоенные каждой записи. Чтобы сделать это, используйте функцию `get_the_term_list()`. Она работает схоже с `get_the_tag_list()`, но предназначена для построения списка элементов пользовательской таксономии.

```
<?php echo get_the_term_list( $post->ID, 'type', 'Product Type: ',
    ', ', '' ); ?>
```

Предыдущий код отображает все элементы пользовательской таксономии, присвоенные просматриваемой вами записи. Его нужно поместить в цикл в файле шаблона темы, чтобы он работал правильно. Для выполнения функции вы посылаете ID записи, имя пользовательской таксономии и название, которое вы хотите отобразить рядом с элементами. Помните, что вы всегда можете обратиться к справке

по функции, чтобы узнать больше о ней и о допустимых параметрах: http://codex.wordpress.org/Function_Reference/get_the_term_list.

Функция `get_terms()` также может использоваться для возврата массива значений пользовательской таксономии. В следующем примере вы возвращаете все элементы для таксономии `type` и прогоняете через цикл значения, отображающие элемент имени:

```
<?php
$terms = get_terms( 'type' );
foreach ( $terms as $term ) {
    echo '<p>' .$term->name. '</p>';
}
?>
```

Помните, что необходимо убедиться в том, что таксономия определена, перед тем как начинать работать со значениями пользовательской таксономии. Если какой-либо из предыдущих примеров не возвращает ничего, это значит, что код выполняется до вызова функции `register_taxonomy()`, определяющей вашу пользовательскую таксономию.

Определение пользовательской таксономии в WordPress — мощный метод организации контента веб-сайта. Использование названных выше методов может помочь превратить сайт в систему управления контентом благодаря мощности WordPress.

Метаданные

С помощью этой главы вы научились создавать пользовательские типы записей, чтобы добавлять их к базовым типам контента, управляемого WordPress, и пользовательские таксономии, чтобы организовывать и собирать эти типы контента. Эта глава завершается рассмотрением возможности расширить дескрипторы управления контентом записи с помощью пользовательских метаданных.

Что такое метаданные?

В WordPress под метаданными подразумеваются дополнительные фрагменты информации, прикрепленные к записи. Например, пользовательскому типу записей `products` может потребоваться цена, хранящаяся вместе с каждым имеющимся товаром. Цена может храниться в виде метаданных и легко отображаться на странице с подробной информацией о товаре.

Метаданные в терминологии WordPress часто называют произвольными (пользовательскими) полями (Custom Fields). Это более понятный пользователю термин, используемый в консоли администратора. WordPress добавляет метаполе произвольных полей на экран редактирования записи по умолчанию, как показано на рис. 7.6. Если у пользовательского типа записи есть значение `custom-fields`, определенное для аргумента `supports`, это метаполе также появится.

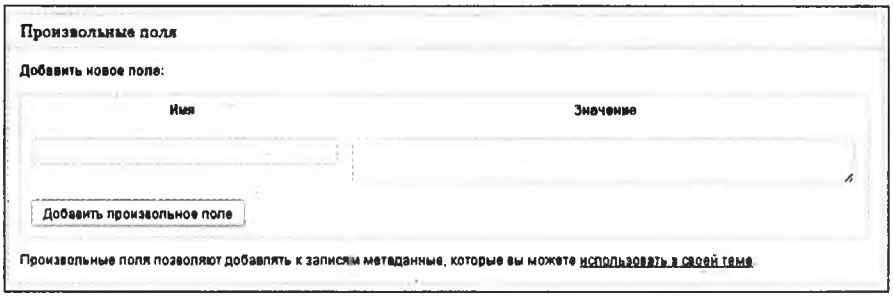


Рис. 7.6. Метаполе произвольного поля

Все метаданные хранятся в таблице `wp_postmeta` в базе данных WordPress.

Добавление метаданных

WordPress поддерживает простую функцию для добавления новых метаданных к записи под названием `add_post_meta()`. Эта функция прикрепляет фрагмент метаданных к определенной записи следующим образом:

```
<?php add_post_meta( $post_id, $meta_key, $meta_value, $unique ); ?>
```

Функция принимает четыре параметра:

- `$post_id` — ID записи, к которой добавляются метаданные.
- `$meta_key` — имя поля метаданных.
- `$meta_value` — значение поля метаданных.
- `$unique` — значение, определяющее, будет ли ключ уникальным. По умолчанию `false`.

Теперь, когда вы знаете параметры функции `add_post_meta()`, вы можете использовать ее для добавления метаданных к товарам.

```
add_post_meta( 420, 'prowp_price', '34.99', true );
```

Этот пример кода добавляет метаданные с названием `prowp_price` и значением 34.99 к товару ID 420. Вы также задаете `$unique` значение `true`, это означает, что для поля `prowp_price` данного товара не может быть множественной информации. Теперь, если вы редактируете товар в WordPress, вы увидите поле `prowp_price` и его значение в метаполе произвольных полей.

ЗАМЕЧАНИЕ

Чтобы ключи метаданных не появлялись в метаполе произвольных полей на экране редактирования записи, добавьте к метаключу префикс с нижним подчеркиванием типа `_prowp_price`. Это скроет данные от пользователей, что является обычной практикой при создании произвольных метаполей.

Обновление метаданных

С той же легкостью, с какой вы добавляете метаданные к записи, вы также можете обновлять их, используя функцию `update_post_meta()`. Эта функция обновит фрагмент метаданных, прикрепленный к выбранной записи, как это показано здесь. Если метаключа еще не существует, функция создаст его.

```
<?php update_post_meta( $post_id, $meta_key, $meta_value, $prev_value ); ?>
```

Функция принимает следующие параметры:

- `$post_id` — ID записи, для которой обновляются метаданные.
- `$meta_key` — имя поля метаданных.
- `$meta_value` — значение поля метаданных.
- `$prev_value` — старое значение поля метаданных для обновления. Существует для различения нескольких полей с одинаковым ключом, является необязательным.

Например, вы можете обновить цену товара:

```
update_post_meta( 420, 'prowp_price', '6.99' );
```

Пример кода обновляет ранее добавленное поле метаданных `prowp_price` на 6.99 для товара ID 420.

Удаление метаданных

Теперь, когда вы знаете, как добавлять и обновлять метаданные записи, вы можете научиться их удалять. Чтобы удалить метаданные записи, используйте функцию `delete_post_meta()`.

```
<?php delete_post_meta( $post_id, $meta_key, $meta_value ); ?>
```

Функция принимает следующие параметры:

- `$post_id` — ID записи, откуда удаляются метаданные.
- `$meta_key` — имя поля метаданных.
- `$meta_value` — значение поля метаданных. Существует для различения нескольких полей с одинаковым ключом, является необязательным.

Давайте удалим метаданные, созданные ранее:

```
delete_post_meta( 420, 'prowp_price' );
```

Пример кода удаляет метаданные `prowp_price` для товара с ID 420. Вы не определили параметр `$meta_value`, поэтому все записи `prowp_price` для товара с ID 420 будут удалены.

Возвращение метаданных

Вы выяснили, как добавлять, обновлять и удалять метаданные. Теперь можно посмотреть, как их возвращать и отображать. WordPress делает простым получение метаданных для отображения или использования в другом коде. Хорошо использовать этот код внутри цикла, чтобы отображать индивидуальные метаданные для определенного фрагмента содержания.

Чтобы вернуть метаданные, используйте функцию `get_post_meta()`:

```
<?php $meta_values = get_post_meta( $post_id, $key, $single ); ?>
```

Функция принимает следующие параметры:

- `$post_id` — ID записи, для которой возвращаются метаданные.
- `$meta_key` — имя поля метаданных.
- `$single` — значение, определяющее, возвращать одиночное значение (`true`) или же массив значений (`false`). По умолчанию `false`.

Давайте вернем и отобразим цену товара, созданного ранее:

```
$product_price = get_post_meta( 420, 'prowp_price', true );
echo 'Price $' .$product_price;
```

Цена товара возвращена и отображается для товара с ID 420. Теперь предположим, что вы хотите добавить для товара описание цвета. Вместо создания отдельных метаданных для каждого цвета вы создадите массив цветов в одиночном поле метаданных:

```
<?php
add_post_meta( 420, 'prowp_colors', 'orange', false );
add_post_meta( 420, 'prowp_colors', 'black', false );
$product_colors = get_post_meta( 420, 'prowp_colors', false );
echo '<ul class="product-colors">';
foreach ( $product_colors as $color ) {
    echo '<li>' .$color .'</li>';
}
echo '</ul>';
?>
```

Сначала вам необходимо создать метаданные для цветов товара. Это делается с помощью функции `add_post_meta()`. Затем установите одинаковое имя метаданных и задайте параметру `$unique` значение `false`, что позволит делать много записей с одним метаключом.

Затем используйте функцию `get_post_meta()` для возврата только что заданных цветов товара. Обратите внимание, что параметр `$single` имеет значение `false`, что позволяет возвращать все записи для `prowp_colors` для товара с ID 420 как массив. Наконец, прогоните цикл через массив цветов и отобразите каждый цвет товара.

Другой мощной функцией для возврата метаданных записи является `get_post_custom()`. Эта функция возвращает многомерный массив всех метаданных для определенной записи.

```
<?php get_post_custom( $post_id ); ?>
```

Она принимает единственный обязательный параметр — `$post_id` — ID записи, произвольные поля которой запрашиваются.

Давайте вернем и отобразим все метаданные для товара:

```
<?php
$product_metadata = get_post_custom( 420 );
foreach( $product_metadata as $name => $value ) {
    echo '<strong>' .$name .'</strong>  =>  ';
    foreach( $value as $nameAr => $valueAr ) {
        echo '<br />' .$nameAr."  =>  ";
        echo var_dump( $valueAr );
    }
    echo '<br />';
}
?>
```

Пример кода возвращает все метаданные для товара с ID 420. Поскольку вернувшееся значение является многомерным массивом, вам нужно прогнать несколько циклов для отображения всех данных. Если вы возвращаете множество фрагментов метаданных для записи, это оптимизированный метод, поскольку он возвращает все метаданные в один запрос базы данных вместо нескольких отдельных запросов для каждого запрошенного фрагмента. Это более продвинутый метод получения метаданных.

Резюме

Легко заметить, что комбинирование пользовательских типов записей, пользовательских таксономий и метаданных WordPress ведет к бесконечному числу возможностей. Именно это позволило WordPress превратиться из простой платформы для блогов в полноценную систему управления контентом, способную работать с любыми типами предоставленных данных.

В следующей главе вы углубитесь в создание пользовательских плагинов для WordPress. Вы научитесь правильным образом интегрировать их в различные области WordPress, поймете, как проводить валидацию данных для разработки безопасного кода, и даже узнаете, как публиковать свои плагины в директории плагинов на WordPress.org.

8 Разработка плагинов

В этой главе:

- Создание файлов плагина
- Валидация данных и безопасность плагина
- Использование фильтра WordPress и зацепок-действий
- Как правильно использовать настройки API
- Создание виджета и консольного виджета
- Создание произвольных сокращенных кодов
- Поддержка языковых переводов
- Публикация плагина в официальной директории плагинов

Загрузка кодов для этой части с Wrox.com

Код для этой части можно скачать по ссылке www.wrox.com/remtitle.cgi?isbn=9781118442272 во вкладке Скачать код (Download Code). Код содержится в загрузках для главы 8 и назван в соответствии с именами файлов кода, используемыми в этой главе.

Одна из основных причин популярности WordPress как программной платформы — легкость его расширения. Плагины — первооснова этой особенности, они дают бесконечные возможности расширения WordPress. В этой главе обсуждается всё, что необходимо для создания интересных плагинов для WordPress.

Вы рассмотрите плагины как с функциональной, так и со структурной точки зрения. Начав с компоновки файлов плагина, вы изучите зацепки API, соединяющие код произвольного плагина с ядром WordPress, и посмотрите, как интегрировать плагин в различные части процессов редактирования, управления и отображения в WordPress. Наконец, вы узнаете, как опубликовать плагин, чтобы его могли использовать другие пользователи. В конце главы вы создадите плагин с нуля.

Вы используете различные возможности, рассмотренные в этой главе, и освоите правильный метод расширения WordPress с помощью произвольных плагинов.

Компоновка плагина

При разработке плагина в WordPress лучше следовать стандартному шаблону компоновки плагина, то есть определенному набору функциональных и описательных компонентов, который существует во всех плагинах, создаваемых для WordPress. В этой главе обсуждаются требования к плагину, а также рекомендованные дополнения, такие как лицензия на программное обеспечение и интернационализация. В то время как реализация плагина является самой увлекательной частью процесса, компоновка плагина схожа с простейшими грамматическими правилами нового языка: это необходимость, чтобы быть понятным.

Создание файла плагина

Первый шаг в создании плагина WordPress — это создание нового PHP-файла для кода плагина. Имя файла плагина должно быть описательным, чтобы его легко можно было идентифицировать в директории `plugins`. Оно также должно быть уникальным, поскольку все плагины WordPress находятся в одной и той же папке. Если имя плагина будет слишком типичным, вы рискуете обнаружить еще один файл с таким именем, что, очевидно, станет проблемой.

Плагин также может находиться в папке, содержащей все файлы, необходимые для работы плагина. Папки для отдельных плагинов лучше использовать всегда, поскольку это помогает держать пользовательскую папку плагинов в порядке. Также неплохо поддерживать ясную структуру этой папки, то есть хранить схожие файлы вместе. Например, если плагин содержит изображения, нужно создать папку `/images` внутри папки плагина для хранения любых произвольных изображений, которые может использовать плагин.

Давайте рассмотрим стандартную структуру папок для плагина:

- `/unique-plugin-name` (без пробелов и специальных символов)
  - `unique-plugin-name.php` (основной PHP-файл плагина)
  - `uninstall.php` (файл отмены установки плагина)
  - `/js` (папка для файлов JavaScript)
  - `/css` (папка для файлов каскадного стиля)
  - `/includes` (папка для включений PHP)
  - `/images` (папка для изображений плагина)

Упорядочивание файлов с помощью четкой структуры папок облегчает отслеживание работы плагина.

Создание заголовка плагина

Требование ко всем плагинам WordPress — действительный заголовок плагина. Он должен быть определен в самом верху основного файла PHP как комментарий PHP. Он не должен находиться во всех файлах плагина, только в основном. Заголовок сообщает WordPress, что файл PHP является законным плагином WordPress и должен обрабатываться соответствующим образом. Вот пример заголовка стандартного плагина:

```
<?php
/*
Plugin Name: Halloween Plugin
Plugin URI: http://example.com/wordpress-plugins/halloween-plugin
Description: This is a brief description of my plugin
Version: 1.0
Author: Michael Myers
Author URI: http://example.com
License: GPLv2
*/
?>
```

Единственная обязательная строка в заголовке плагина — это `Plugin Name`. Остальная информация необязательна, но ее наличие настоятельно рекомендуется. Информация, содержащаяся в заголовке плагина, используется в разделе «Управление плагинами» WordPress. Вы можете посмотреть, на что похож заголовок, на рис. 8.1.

Рис. 8.1. Пример листинга плагина

Вы видите, как важна информация в заголовке плагина, включая все дополнительные данные. Информация должна быть точной, предоставлять правильные ссылки на веб-сайт и URL плагина для дополнительной информации и поддержки.

Лицензия плагина

При разработке плагина, который вы планируете опубликовать, можно включить лицензию на это программное обеспечение в его заголовок. Это не является требованием для функционирования плагина, но полезно для внесения ясности по используемой плагином лицензии. Блок комментария лицензии содержит информацию о том, что вы не предоставляете никаких гарантий. Это поможет, если кто-либо решит предъявить вам претензии, что ваш плагин разрушил его сайт. Ниже приведена стандартная лицензия GPL, с которой выпускается большинство плагинов WordPress:

```
<?php
/*  Copyright YEAR  PLUGIN_AUTHOR_NAME  (email : PLUGIN AUTHOR EMAIL)
    This program is free software; you can redistribute it and/or modify
```

```
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.
    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/
?>
```

Чтобы использовать эту лицензию для вашего плагина, внесите в предыдущий комментарий год, имя и электронный адрес автора. После этого ваш плагин будет лицензирован в соответствии GPL.

WordPress лицензирован в соответствии с лицензией на программное обеспечение GPLv2. Это распространенный вариант для проектов с открытым исходным кодом. Поскольку работа плагинов зависит от WordPress, они также должны лицензироваться по GPL или совместимому варианту. Более подробная информация о GPL содержится по ссылке http://www.gnu.org/licenses/licenses.html.

Функции активации и деактивации

При создании новых плагинов вам понадобится использовать несколько важных функций. Первая называется `register_activation_hook()`. Эта функция выполняется при активации плагина во вкладке Плагины WordPress. Она принимает два параметра: путь к основному файлу плагина и функцию для выполнения после его активации.

В большинстве примеров кода, приведенных в этой части, вы будете использовать `prowp` в качестве функции и переменный префикс, а также описательное имя для плагина. Это просто краткое название примера, но вы его будете частенько встречать в коде. В приведенном ниже примере при активации плагина исполняется функция `prowp_install()`:

```
<?php
register_activation_hook( __FILE__, 'prowp_install' );
function prowp_install() {
    // делаем то, что нужно
}
?>
```

Это крайне полезная функция для выполнения любых действий после активации плагина. Например, вам может захотеться проверить текущую версию WordPress, чтобы убедиться, что плагин совместим с ней. Вам также может понадобиться создать несколько настроек параметров по умолчанию.

Одна из проверок, которую необходимо проводить при активации плагина, — это проверка совместимости используемой версии WordPress с запущенным плагином.

Это гарантирует, что любые функции, зацепки и прочие возможности, используемые плагином, доступны в WordPress.

```
<?php
register_activation_hook( __FILE__, 'prowp_install' );
function prowp_install() {
    global $wp_version;
    if ( version_compare( $wp_version, '3.5', '<' ) ) {
        wp_die( 'This plugin requires WordPress version 3.5 or higher.' );
    }
}
?>
```

Приведенная выше функция использует глобальную переменную `$wp_version`, которая хранит информацию об используемой в данный момент версии WordPress и подтверждает, что она не ниже версии 3.5. Сравнение версий осуществляется с помощью функции PHP `version_compare()`. Если версия WordPress ниже 3.5, будет отображено сообщение о необходимости обновления.

Есть также и функция, которая выполняется, когда плагин деактивирован. Это `register_deactivation_hook()`. Она запускается, когда плагин деактивирован во вкладке Плагины WordPress. Функция принимает те же два аргумента, что и `register_activation_hook`. Вот пример ее использования:

```
<?php
register_deactivation_hook( __FILE__, 'prowp_deactivate()' );
function prowp_deactivate() {
    // делаем то, что нужно
}
?>
```

ЗАМЕЧАНИЕ

Важно помнить, что деактивация не является деинсталляцией. В функцию деактивации никогда не следует включать возможность деинсталляции. Представьте себе, что пользователь случайно деактивирует плагин, удаляя при этом все настройки. Такого «приятного» пользовательского опыта следует избегать.

Интернационализация

Интернационализация, которую в кодексе WordPress сокращают до «i18n», — это процесс подготовки плагина или темы к переводу или локализации. В WordPress это означает наличие строк, которые необходимо перевести. Локализация — это процесс перевода текста, отображаемого темой или плагином, на различные языки. Это не обязательное требование, но интернационализацию желательно использовать для любого плагина, который планируется к распространению. Таким образом он будет доступен для максимально широкой аудитории.

WordPress поддерживает множество различных функций, чтобы сделать строку пригодной для перевода. Первая — это `__()`. Это не опечатка: функция использует два нижних подчеркивания, как показано в примере:

```
<?php $howdy = __( 'Howdy Neighbor!', 'prowp-plugin' ); ?>
```

Первый передаваемый параметр — это строка, которую следует перевести. Она будет отображаться в браузере, если текст не переведен на другой язык. Второй параметр — это область текста. В случае тем и плагинов область должна быть уникальным идентификатором, используемым для различения между всеми загруженными переводами.

Если код должен отражать переводимую строку в браузере, вам понадобится использовать функцию `_e()`:

```
<?php _e( 'Howdy Neighbor!', 'prowp-plugin' ); ?>
```

Она работает точно так же, как функция `__()`; разница в том, что она отражает значение в браузере.

При интернационализации плагинов и тем следует уделять особое внимание «заполнителям». Например, посмотрим на сообщение об ошибке, которое мы хотели бы сделать переводимым:

```
Error Code 6980: Email is a required field
```

очевидный, но неверный путь разделить строку на переводимые части — это отделить имя поля, номер ошибки и строку с описанием:

```
<?php
$error_number = 6980;
$error_field = "Email";
$error = __( 'Error Code ', 'prowp-plugin' ) .$error_number. ': '
.$error_field .__( ' is a required field', 'prowp-plugin' );
echo $error;
?>
```

Неверно включать динамические значения в переводимую строку, поскольку она разделяется на две части. Эти две части могут не быть независимыми в другом языке. А это может серьезно смутить переводчика, просматривающего криптографированные фразы, ничего не значащие в случае их разбиения. Вот правильный метод:

```
<?php
$error_number = 6980;
$error_field = "Email";
printf( __( 'Error Code %1$d: %2$s is a required field', 'prowp-plugin' ),
$error_number, $error_field );
?>
```

Как вы видите, здесь используется функция PHP `printf()`, которая выдает отформатированную строку. Обе переменные передаются `printf()` и включаются в строку в предназначенных для этого местах. В данном примере разработчик, переводящий сообщения плагина на другой язык, будет видеть строку как `Error Code %1$d: %2$s is a required field` и знать, что можно переставлять слова вокруг номера ошибки и значений поля, чтобы получить осмысленную фразу в конечном языке. Разбиение строк ведет к разбиению перевода и неожиданно комичным

грамматически результатам. В качестве альтернативы можно использовать функцию PHP `sprintf()`, если вы хотите хранить значение сообщения об ошибке в переменной до его отображения.

Особое внимание при определении переводимых строк следует обращать и на множественные числа. Скажем, вам нужно перевести строку следующего типа:

```
<?php
$count = 1;
printf( __( 'You have %d new message', 'prowp-plugin' ), $count );
?>
```

Все отлично, если у вас только одно новое сообщение. Но что, если их больше? К счастью, в WordPress есть функция, которую можно использовать для решения этой проблемы: `_n()`. Вот как она работает:

```
<?php
$count = 34;
printf( _n( 'You have %d new message', 'You have %d new messages',
$count, 'prowp-plugin'), $count );
?>
```

Функция принимает четыре параметра: единственное число, множественное число, реальное число и область текста для плагина. Функция `_n()` использует численный параметр (`$count` в данном примере), чтобы определить, строка с единственным или с множественным числом должна быть возвращена.

WordPress также поддерживает функцию перевода, которую вы можете использовать для добавления комментариев к переводимым строкам. Это полезно, если у вас есть предназначенная для перевода строка, у которой может быть несколько значений. Вот пример использования функции `_x()`:

```
<?php
echo _x( 'Editor', 'user role', 'prowp-plugin' );
echo _x( 'Editor', 'rich-text editor', 'prowp-plugin' );
?>
```

Как вы видите, функция принимает три параметра. Первый — текстовая строка для перевода. Второй, самый важный, — контекстная информация для переводчиков. Этот параметр позволяет добавлять произвольные комментарии, объясняющие переводчику контекст переводимого текста. Последний параметр — область текста.

После подготовки плагина к переводу необходимо загрузить файл локализации, чтобы этот перевод выполнить. Чтобы сделать это, запустите функцию `load_plugin_textdomain()`:

```
<?php
add_action( 'init', 'prowp_init' );
function prowp_init() {
    load_plugin_textdomain( 'prowp-plugin', false,
    plugin_basename( dirname( __FILE__ ) .'/localization' ) );
}
?>
```

Первый передаваемый параметр — это имя области текста, которое вы использовали для идентификации всех переводимых строк. Второй параметр — путь относительно переменной ABSPATH; однако этот параметр сейчас считается устаревшим (deprecated), поэтому рекомендуется использовать третий. Последний параметр — путь к файлам перевода из директории /plugins. Чтобы хранить эти файлы, нужно создать папку с названием /localization внутри директории плагина. Используйте функции plugin_basename() и dirname(), чтобы вернуть путь к папке локализации.

О процессе создания файлов перевода можно узнать больше в Кодексе WordPress http://codex.wordpress.org/I18n_for_WordPress_Developers.

Определение путей

При создании плагинов WordPress вам нередко будет нужно обращаться к файлам и папкам внутри копии WordPress. Начиная с версии WordPress 2.6 у пользователей есть возможность перемещать эту директорию куда угодно. Поэтому никогда не следует задавать фиксированные пути внутри плагина. В WordPress есть набор функций для определения пути к директориям wp-content и plugins, а также к директориям внутри плагинов. Вы можете использовать эти функции в своих плагинах, чтобы быть уверенными, что любые пути, которые вы используете, верны, вне зависимости от того, где именно на сервере располагается директория.

Локальные пути

Чтобы определить путь к плагину на локальном сервере, используйте функцию plugin_dir_path(). Эта функция извлекает физическое положение относительно директории плагинов из имени файла.

```
<?php echo plugin_dir_path( __FILE__ ); ?>
```

Вы видите, что PHP-постоянная __FILE__ передается функции plugin_dir_path(). Она возвращает полный путь к директории плагина на локальном сервере:

```
/public_html/wp-content/plugins/halloween-plugin/
```

Теперь предположим, что вам нужен локальный путь к файлу в поддиректории в плагине. Вы также можете использовать функцию plugin_dir_path() для поддиректорий и файлов, на которые необходимо сослаться:

```
<?php echo plugin_dir_path( __FILE__ .'js/script.js' ); ?>
```

Предыдущий пример даст следующий результат:

```
/public_html/wp-content/plugins/halloween-plugin/js/script.js
```

Пути URL

Чтобы определить полный URL к любому файлу в директории плагина, используйте функцию plugins_url():

```
<?php echo '<img src="' .plugins_url( 'images/icon.png', __FILE__ ). '">'; ?>
```

Видно, что функция `plugins_url()` принимает два параметра. Первый параметр — это путь относительно URL плагина, второй — файл плагина, на который вы хотите сослаться. В данном случае — PHP-константа `__FILE__`. Предшествующий пример вернет полный URL для файла плагина `icon.png`, расположенного в директории изображений:

```
<img src="http://example.com/wp-content/plugins/halloween-plugin/images/icon.png">
```

Ниже приведен список основных преимуществ функции `plugins_url()` для определения URL файлов плагина:

- поддерживает директорию `/mu-plugins`;
- автоматически определяет SSL. Если SSL активирован, возвращаемый URL будет содержать `https://`;
- может определить положение плагина, даже если пользователь переместил директорию `/wp-content` в произвольное местоположение;
- поддерживает Multisite.

WordPress снабжен различными функциями для определения URL:

- `admin_url()` — администраторский URL (http://example.com/wp-admin/).
- `site_url()` — URL сайта для текущего сайта (http://example.com).
- `home_url()` — домашний URL текущего сайта (http://example.com).
- `includes_url()` — URL директории Includes (http://example.com/wp-includes/).
- `content_url()` — URL директории Content (http://example.com/wp-content/).
- `wp_upload_dir()` — возвращает массив с локальной информацией по сконфигурированной директории загрузок.

Понимание правильного способа доступа к файлам в плагинах принципиально для гарантии максимальной совместимости со всеми копиями WordPress, вне зависимости от их настроек.

Безопасность плагина

Один из самых важных шагов в создании плагина — убедиться в его защищенности от взломов. Если плагин содержит прорехи в безопасности, он делает весь сайт на WordPress открытым для тотального опустошения злобными хакерами. WordPress снабжен встроенными инструментами безопасности, которые всегда следует использовать, чтобы быть уверенными в том, что плагины безопасны настолько, насколько это в принципе возможно.

Помните, что все внешние данные для кода плагина находятся под подозрением, пока не будут проверены. Всегда проверяйте данные перед тем, как отображать их

в браузере или вставлять в базу данных. Это поможет вам защищать плагины от взломов. Вам понадобится использовать функции исключения и очистки, которые мы обсудим в этой главе.

Временные значения (Nonces)

Nonces, что означает «число, использованное единожды», применяются в запросах (сохранение параметров, формы записей, запросы Ajax, действия), чтобы остановить неавторизованный доступ посредством генерирования секретного ключа. Секретный ключ генерируется перед созданием запроса (например, формы записи). После этого ключ передается от запроса к скрипту и проверяется на соответствие до начала какой-либо обработки. Теперь давайте посмотрим, как создавать и проверять временные значения вручную. В следующем примере временное значение используется в форме:

```
<form method="post">
    <?php wp_nonce_field( 'prowp_settings_form_save', 'prowp_nonce_field' ); ?>
    Enter your name: <input type="text" name="text" /><br />
    <input type="submit" name="submit" value="Save Options" />
</form>
```

При создании временного значения формы функцию `wp_nonce_field()` следует вызывать внутри тегов `<form>`. Обязательных параметров для функции нет, но для повышения уровня безопасности следует установить два параметра. Первый, `$action`, должен быть уникальной строкой, являющейся описанием совершаемого действия. Второй — уникальное имя поля, `$name`. По умолчанию именем поля является `_wpnonce`, но вы можете определить произвольное имя в этом параметре.

При вызове функция `wp_nonce_field()` генерирует уникальный секретный ключ, который добавляется как скрытое поле формы и передается с данными формы. После публикации формы прежде всего следует проверить, что секретный ключ с временным значением, использующий функцию `check_admin_referer()`, функционирует следующим образом:

```
function prowp_update_options() {
    if ( isset( $_POST['submit'] ) ) {
        // проверяем временное значение из соображений безопасности
        check_admin_referer( 'prowp_settings_form_save', 'prowp_nonce_field' );
        // проверка произведена, теперь делаем то, что нужно
    }
}
```

Для проверки действительности временного значения достаточно вызвать функцию `check_admin_referer()` и передать ей уникальное действие временного значения и название, определенные ранее. Если секретный ключ с временным значением не соответствует секретному ключу, созданному в форме, WordPress остановит обработку страницы и выдаст сообщение об ошибке. Это обеспечивает первичную защиту от подделки межсайтовых запросов (CSRF, cross-site request forgery).

Временные значения также могут использоваться в ссылках, которые обеспечивают действия. Чтобы создать временное значение для URL, используйте функцию `wp_nonce_url()`. Ее можно применять вместе с множественными строками запросов в URL:

```
<?php
$link = 'my-url.php?action=delete&ID=15';
?>
<a href="<?php echo wp_nonce_url( $link, 'prowp_nonce_url_check' ); ?>">Delete</a>
```

Функция `wp_nonce_url()` принимает два параметра: URL, к которому следует добавить временное значение, и имя уникального временного значения, которое вы создаете. Предшествующий код сгенерирует ссылку такого типа:

```
http://example.com/wp-admin/my-url.php?action=delete&ID=15&_wpnonce=e9d6673015
```

Обратите внимание, что строка запроса `_wpnonce` прикрепляется в конце ссылки. Это значение секретного ключа, которое было сгенерировано для временного значения URL. Если у URL нет строк запросов, функция `wp_nonce_url()` добавит временное значение, как при передаче единственной строки запроса. Если URL содержит несколько строк запроса, значение временного параметра будет добавлено в конце URL. Вы можете проверить правильность временного значения так же, как и в случае с формой, — используя функцию `check_admin_referer()`:

```
function prowp_update_options() {
    if ( isset( $_GET['action'] ) ) {
        // проверяем временное значение из соображений безопасности
        check_admin_referer( 'prowp_nonce_url_check' );
        // делаем то, что нужно
    }
}
```

Эта функция проверяет, что строка запроса действия установлена до проверки временного значения. При подтверждении временного значения выполнение сценария продолжается. Помните, что в случае неподтверждения временного значения загрузка страницы будет остановлена, чтобы предотвратить любые попытки взлома.

Валидация и очистка данных

Любые данные, поступающие в код извне (например, пользовательский ввод), должны быть очищены, чтобы убедиться, что в них нет непригодных символов и потенциально небезопасных данных. Валидация данных принципиально важна для правильной безопасности плагина. Неверно валидированные данные могут привести к инъекциям SQL, взломам, ошибкам и т. д.

WordPress поддерживает набор функций обработки строк, которые можно использовать, чтобы убедиться, что при отображении на экране все данные обработаны и готовы к показу. Эти функции обработки следуют стандарту установки имен

(см. список далее), что позволяет понять, что именно они обрабатывают. На рис. 8.2 показаны шаблоны наименования функций исключения.

- `esc_` — префикс для функций обработки.
- `attr` — контекст (`attr`, `html`, `textarea`, `js`, `sql`, `url`, and `url_raw`).
- `_e` — необязательный суффикс перевода. Доступны суффиксы `__` и `_e`.

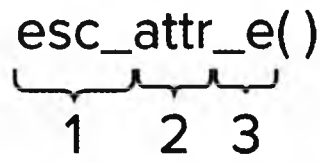


Рис. 8.2. Правила именования функций обработки

Функция `esc_html()` используется для обработки данных, содержащих HTML. Она кодирует специальные символы в эквивалентные сущности HTML. Символы включают в себя &, <, >, " и ':

```
<?php esc_html( $text ); ?>
```

Функция `esc_attr()` используется для обработки атрибутов HTML. Она должна применяться, если вам необходимо отобразить данные внутри элемента HTML:

```
<input type="text" name="first_name" value="<?php echo esc_attr( $text ); ?>">
```

Функция `esc_textrea()` используется для обработки HTML значений `<textarea>`. Она должна использоваться для кодирования текста, включаемого в элемент формы `<textarea>`:

```
<textarea name="description"><?php echo esc_textarea( $text ); ?></textarea>
```

WordPress также поддерживает функцию валидации URL, которая называется `esc_url()`. Ее следует использовать для очистки URL от непригодных символов. Несмотря на то что технически `href` является атрибутом HTML, использовать функцию `esc_url()` следует таким образом:

```
<a href="<?php echo esc_url( $url ); ?>">
```

Функция `esc_js()` обрабатывает текстовые строки на JavaScript:

```
<script>
    var bwar='<?php echo esc_js( $text ); ?>';
</script>
```

Функция `esc_sql()` обрабатывает данные для использования запроса MySQL. На самом деле эта функция — всего лишь сокращенный вариант `$wpdb->escape()`:

```
<?php esc_sql( $sql ); ?>
```

Необязательный суффикс перевода (__ или _e) используется для перевода обработанных данных. Суффикс _e будет отображать переведенный текст, тогда как __ только возвращает переведенное значение.

```
<?php
// обрабатываем, переводим и отображаем текст
esc_html_e( $text, 'prowp-plugin' );
// обрабатываем, переводим, но не отображаем текст
$text = esc_html__( $text, 'prowp-plugin' );
?>
```

Если данные, которые вы подтверждаете, должны быть целыми, то для подтверждения этого факта используйте PHP-функцию `intval()`. Функция `intval()` вернет целое значение переменной. Если переменная представляет собой строку, будет возвращен 0.

```
$variable = 12345;
$variable = intval( $variable );
```

Другая полезная функция для работы с целыми — это `absint()`. Она проверяет, что результат — неотрицательное целое:

```
$variable = 12345;
$variable = absint( $variable );
```

WordPress поддерживает некоторые полезные функции очистки. Их следует использовать перед сохранением в базе данных. Одна из таких функций — `sanitize_text_field()`. Она удаляет все недействительные символы UTF-8, конвертирует единичные угловые скобки < в объекты HTML и удаляет все теги, разрывы строки и дополнительные пробелы.

```
<?php sanitize_text_field( $text ); ?>
```

Вы также можете очистить электронный адрес, используя `sanitize_email()`. Эта функция выберет все символы, недопустимые в электронном адресе. Рассмотрим следующий код:

```
<?php
$sanitized_email = sanitize_email( '        éric@loremipsum.com!' );
echo $sanitized_email; // отобразит: ric@loremipsum.com
?>
```

Вы видите, что функция `sanitize_email()` удаляет все дополнительные пробелы и непригодные символы из присланного электронного адреса.

Мощной функцией обработки и очистки ненадежного HTML является `wp_kses()`. Она используется в WordPress, чтобы проверить, что пользователи посылают только разрешенные теги и атрибуты HTML. Определяя разрешенные теги HTML, вы можете избежать атак посредством межсайтингового скриптинга (XSS) через ваш код. Рассмотрим пример:

```
$allowed_tags = array(
    'strong'    =>    array(),
    'a'         =>    array(
```

```
        'href'         =>    array(),
        'title'        =>    array()
    )
);
$html = '<a href="#" class="external">link</a>.
    This is <b>bold</b> and <strong>strong</strong>';
echo wp_kses( $html, $allowed_tags );
```

Первый шаг — определение массива всех тегов и атрибутов HTML. В данном примере вы разрешаете теги `<strong>` и `<a>`. Тег `<a>` позволяется для включения атрибутов `href` и `title`. Затем вы строите переменную `$html` для проверки функции. Последний шаг — передача строки `$html` и аргументов `$allowed_tags` функции `wp_kses()`.

Предыдущий пример отобразит следующий код:

```
<a href="#">link</a>.  This is bold and <strong>strong</strong>
```

Обратите внимание, что теги `<b></b>` полностью удаляются. Функция также удаляет атрибут `class` из тега `<a>`, поскольку вы не определили его как допустимый. Этот базовый пример показывают реальную силу данной функции. В любое время, когда вам необходимо позволить пользователям вставить код HTML, следует использовать функцию `wp_kses()` для проверки того, что разрешены только допустимые теги и атрибуты HTML.

Чтобы узнать больше о валидации данных в WordPress, обратитесь к следующей статье Кодекса: http://codex.wordpress.org/Data_Validation.

ЗАМЕЧАНИЕ

На протяжении этой главы вы будете использовать различные техники валидации данных в примерах кода. Цель — подчеркнуть важность поддержания безопасности на сайте при разработке плагинов для WordPress.

Знай свои зацепки: действия и фильтры

Одной из наиболее важных возможностей расширения WordPress является *зацепка* (*hook*). Зацепки — это просто стандартизированный способ внесения изменений в WordPress. Используя зацепки, в определенное время в ходе процесса WordPress вы можете выполнять функции, позволяющие определять, как функционирует WordPress и что будет на выходе. Зацепки — первичный способ взаимодействия плагинов с контентом WordPress. До этого момента вы сосредоточивались на структуре и формате плагинов, но теперь пора заставить плагин сделать что-нибудь!

Зацепка — это просто вызов функции PHP с различными возможными параметрами. Ниже приведен пример, показывающий правильно сформатированный вызов зацепки действия:

```
<?php add_action( $tag, $function_to_add, $priority, $accepted_args ); ?>
```

Действия и фильтры

Могут использоваться два типа зацепок: действия и фильтры. Зацепки-действия запускаются событиями в WordPress. Например, зацепка-действие запускается при публикации новой записи. Зацепки-фильтры используются для изменения контента WordPress до его сохранения в базе данных или отображения на экране. Например, зацепка-фильтр доступна для контента записи или страницы. То есть вы можете изменять контент после того, как он был возвращен из базы данных, но до того, как он отображается в браузере.

Посмотрите на пример действия зацепки-фильтра. Помните, что зацепки-фильтры меняют контент, как в этом примере, в котором меняется содержимое записи:

```
<?php add_filter( 'the_content', 'prowp_function' ); ?>
```

Функция `add_filter()` используется для выполнения фильтра-действия. Вы используете фильтр `the_content`, который фильтрует контент записи. Он сообщает WordPress, что каждый раз при отображении контента его необходимо передавать через произвольную функцию `prowp_function()`. Функция `add_filter()` принимает четыре параметра:

1. `filter_action (string)`: используемый фильтр.
2. `custom_filter_function (string)`: произвольная функция для прогона фильтра.
3. `priority (integer)`: приоритет запуска фильтра. В случае прикрепления нескольких функций обратного вызова к одной зацепке параметр `priority` определяет порядок их выполнения.
4. `accepted args (integer)`: количество аргументов, принимаемых функцией.

Вот пример фильтра `the_content` в действии:

```
<?php
add_filter( 'the_content', 'prowp_profanity_filter' );
function prowp_profanity_filter( $content ) {
    $profanities = array( 'sissy', 'dummy' );
    $content= str_ireplace( $profanities, '[censored]', $content );
    return $content;
}
?>
```

Функция `prowp_profanity_filter()` автоматически заменит слова «sissy» и «dummy» на `[censored]` во всех записях и страницах веб-сайта. Чтобы совершить эту замену, вы используете функцию PHP `str_ireplace()`. Она заменит некоторые символы в строке на другие символы в строке. Функция `str_ireplace()` также чувствительна к регистру. Поскольку вы используете зацепку-фильтр, контент не меняется в базе данных, вместо этого он меняется во время обработки `the_post()`, перед отображением, до того как запустится фильтр. Контент базы данных не затрагивается, поэтому слова «sissy» и «dummy» по-прежнему будут существовать в нем, и если вы отключите или измените плагин, они появятся в отображаемом

тексте. Зацепки-фильтры всегда получают данные. В этом случае функции передается переменная `$content`, содержащая контент записи. Также обратите внимание на то, что последняя строка функции возвращает переменную `$content`. Помните, что всегда следует возвращать изменяемое содержимое, иначе оно будет возвращено пустым и в результате не отобразится ничего.

Теперь, когда вы увидели зацепку-фильтр в действии, посмотрим на зацепку-действие и ее возможности. Зацепка-действие запускается событиями в WordPress. WordPress не требует каких-либо возвращаемых значений от функции зацепки-действия, ядро просто сообщает коду, что имело место определенное событие. Зацепка-действие структурирована точно так же, как и зацепка-фильтр, что видно из этого кода:

```
<?php add_action( 'hook_name', 'prowp_function' ); ?>
```

Функция `add_action()` принимает четыре параметра, как и функция `add_filter()`. Вы можете задать имя зацепки, произвольное имя функции, выполняемой при возникновении события, приоритет и число принимаемых аргументов. Вот реальный пример использования зацепки-действия:

```
<?php
add_action( 'comment_post', 'prowp_email_new_comment' );
function prowp_email_new_comment() {
    wp_mail( 'me@example.com', 'New blog comment',
    'There is a new comment on your website: http://example.com' );
}
?>
```

Обратите внимание, что вы используете зацепку-действие `comment_post`. Это действие запускается при появлении в WordPress нового комментария. Как вы видите, функция `prowp_email_new_comment()` посылает новое электронное сообщение всякий раз при создании нового комментария. Обратите также внимание на то, что вы не посылаете функции никаких переменных, а она не возвращает никаких значений. Зацепкам-действиям это не нужно, но при необходимости передать значения в функцию можно.

Популярные зацепки-фильтры

В WordPress есть более 1500 различных зацепок-фильтров, что поначалу ошеломляет. К счастью, лишь немногие используются чаще остальных. Этот раздел охватывает некоторые из наиболее популярных зацепок в WordPress.

Некоторые из популярных зацепок-фильтров:

- `the_content` — применяется к контенту записи или страницы до отображения.
- `the_content_rss` — применяется к контенту записи или страницы для включения RSS.
- `the_title` — применяется к названию записи или страницы до отображения.

- `comment_text` — применяется к тексту комментария до отображения.
- `wp_title` — применяется к странице `<title>` до отображения.
- `the_permalink` — применяется к постоянному URL.

Давайте посмотрим на самые популярные зацепки-фильтры в WordPress начиная с более полезного примера, чем фильтр нецензурной брани, использующего зацепку-фильтр `the_content`. Эта зацепка позволяет вам изменять содержимое записей и страниц до их отображения в браузере. Используя ее, вы можете добавлять произвольный контент до, в середине или после содержимого:

```
<?php
add_filter ( 'the_content', 'prowp_subscriber_footer' );
function prowp_subscriber_footer( $content ) {

    if( is_single() ) {
        $content.= '<h3>Enjoyed this article?</h3>';
        $content.= '<p>Subscribe to my
            <a href="http://example.com/feed">RSS feed</a>!</p>';
    }
    return $content;
}
?>
```

В этом примере добавляется текст для подписки в нижнюю часть контента записей. Обратите внимание, что здесь также используется условный тег `is_single()` для проверки того, что текст для подписки добавляется только на страницу одиночной записи. Переменная `$content` хранит все содержимое записи или страницы, поэтому, добавляя текст для подписки, вы помещаете его в нижнюю часть контента записи. Добавлять контент в нижнюю часть всех записей — идеальный вариант, поскольку вы практически не меняете запись. В будущем, если вы решите изменить это сообщение, вы можете изменить его только в одном месте, а не обновлять каждую запись на сайте.

Другая мощная зацепка-фильтр — это `the_title`. Она используется для изменения названия записи или страницы до ее отображения. Вот пример использования:

```
<?php
add_filter( 'the_title', 'prowp_custom_title' );
function prowp_custom_title( $title ) {
    $title .= ' - By Example.com';
    return $title;
}
?>
```

В этом примере текст `- By Example.com` добавляется к названиям всех записей и страниц. Название в базе данных не изменяется, но меняется его отображение, сгенерированное для конечного пользователя.

Зацепка-фильтр `default_content` пригодится для установки содержимого по умолчанию при создании новой записи или страницы. Это полезно, если у вас есть предустановленный формат для всех записей, поскольку таким образом можно сэкономить время на написание кода:

```
<?php
add_filter( 'default_content', 'prowp_default_content' );
function prowp_default_content( $content ) {
    $content = 'For more great content please subscribe to my RSS feed';
    return $content;
}
?>
```

Зацепки-фильтры исключительно полезны для вставки собственных процессов обработки в разные точки обработки цикла каждой записи. Понять всю силу плагина WordPress — значит также использовать зацепки-фильтры для запуска собственного кода в ответ на события внутри ядра WordPress.

Популярные зацепки-действия

Некоторые из популярных зацепок-действий:

- `publish_post` — срабатывает при публикации новой записи.
- `create_category` — срабатывает при создании новой рубрики.
- `switch_theme` — срабатывает при смене темы.
- `admin_head` — срабатывает в разделе `<head>` в консоли администратора.
- `wp_head` — срабатывает в разделе `<head>` темы.
- `wp_footer` — срабатывает в «подвале» темы, обычно перед тегом `</body>`.
- `init` — срабатывает после того, как WordPress закончил загрузку, но до отправки заголовков. Хорошее место для перехвата HTML запросов `$_GET` и `$_POST`.
- `admin_init` — то же, что и `init`, но запускается только на страницах консоли администратора.
- `user_register` — срабатывает при создании нового пользователя.
- `comment_post` — срабатывает при добавлении нового комментария.

Одна из наиболее часто используемых зацепок-действий — это `wp_head`. Применяя `wp_head`, вы можете вставить любой произвольный код в раздел `<head>` темы WordPress. Рассмотрим следующий пример:

```
<?php
add_action( 'wp_head', 'prowp_custom_css' );
function prowp_custom_css() {
 ?>
    <style type="text/css">
    a {
        font-size: 14px;
        color: #000000;
        text-decoration: none;
    }
    a:hover {
        font-size: 14px
```

```
        color: #FF0000;
        text-decoration: underline;
    }
    </style>
<?php
}
?>
```

Этот код может добавить что угодно внутрь функции `prowp_custom_css()` в заголовке темы WordPress, в данном случае — произвольный сценарий CSS.

Зацепка `wp_footer` также является часто используемой зацепкой-действием. Используя ее, вы можете вставить любой произвольный код в «подвал» темы WordPress. Это прекрасный метод добавления кода аналитического отслеживания на веб-сайт:

```
<?php
add_action( 'wp_footer', 'prowp_site_analytics' );
function prowp_site_analytics()  {
?>
    <script type="text/javascript">
    var gaJsHost = (("https:" == document.location.protocol) ?
      "https://ssl." : "http://www.");
    document.write(unescape("%3Cscript src='" + gaJsHost +
      'google-analytics.com/ga.js' type='text/javascript'%3E%3C/script%3E"));
    </script>
    <script type="text/javascript">
    var pageTracker = _gat._getTracker("UA-XXXXXX-XX");
    pageTracker._trackPageview();
    </script>
<?php
}
?>
```

В предыдущем примере вы увидели, как просто вставить код отслеживания Google Analytics в «подвал» на каждой странице сайта.

Зацепка-действие `admin_head` схожа с зацепкой `wp_head`, но перехватывает не в заголовке темы, а в заголовке консоли администратора. Это полезно, если плагин требует произвольный CSS в консоли администратора или любой другой код заголовка.

Зацепка-действие `user_register` выполняется при создании в WordPress нового пользователя. Пользователь может быть создан администратором или новым пользователем. Это полезная зацепка, если вы хотите установить для нового пользователя какие-то значения по умолчанию или отправить сообщение новому члену сообщества, чтобы поблагодарить его за то, что он присоединился к вашему сайту.

Зацепки, пожалуй, одна из самых слабо задокументированных возможностей WordPress. Найти подходящие зацепки для использования в работе может быть настоящим испытанием. Первый ресурс — это всегда Кодекс. Здесь вы найдете разделы по фильтрам (http://codex.wordpress.org/Plugin_API/Filter_Reference) и действиям

(http://codex.wordpress.org/Plugin_API/Action_Reference), полезные при поиске подходящих зацепок.

Другим рекомендуемым справочником является директория плагинов (http://wordpress.org/extend/plugins/) на WordPress.org. Иногда лучший способ выяснить что бы то ни было — это посмотреть, как другие разработчики справились с аналогичной задачей. Найдите в директории плагин, функциональность которого схожа с тем, что вы хотите создать. Скорее всего, автор плагина уже применил подходящие WordPress зацепки, которые вы сможете использовать. Никогда не стыдно учиться на примерах, а опубликованные плагины — прекрасные примеры в данном случае.

Настройки плагина

У большинства плагинов есть страница настроек. Это помогает пользователю сконфигурировать плагин на работу тем или иным способом, не меняя его код, а только сохраняя различные настройки параметров. Первый шаг в данном процессе — сохранение и возврат параметров в WordPress.

Сохранение параметров плагина

Скорее всего, при создании плагина вам потребуется сохранить для него некоторые параметры. WordPress поддерживает ряд простых в использовании функций для параметров сохранения, редактирования и удаления. Для создания параметров доступны две функции: `add_option()` и `update_option()`. Обе создают параметры, но `update_option()` также обновляет параметры, если они уже существуют. Вот пример добавления нового параметра:

```
<?php add_option( 'prowp_display_mode', 'Fright Night' ); ?>
```

Первый параметр, который вы посылаете функции `add_option()`, — это имя вашего параметра. Это обязательное поле, которое должно быть уникальным и отличаться от других параметров, сохраненных в WordPress, в том числе и от параметров других плагинов. Второй параметр — это значение параметра. Это также обязательное поле, которое может быть строкой, массивом, объектом или сериализованным значением. Для создания новых параметров можно использовать `update_option()`. Эта функция проверяет, существует ли параметр, и создает его в случае отсутствия. Однако если параметр существует, значения будут обновлены на новые, посланные вами. Функция `update_option()` вызывается точно так же, как и в случае добавления параметра:

```
<?php update_option( 'prowp_display_mode', 'Fright Night' ); ?>
```

Обычно для добавления и обновления параметров плагина используется функция `update_option()`. Гораздо проще вызывать одну функцию для двух задач, чем несколько различных для добавления и обновления параметров плагина.

Получить значение параметра так же просто. Чтобы получить любой параметр, используйте функцию `get_option()`:

```
<?php echo get_option( 'prowp_display_mode' ); ?>
```

Единственное обязательное поле для `get_option()` — это имя параметра, который вы хотите вернуть. Если параметра не существует, функция возвращает `FALSE`.

Параметры можно удалить так же легко, как они были созданы. Чтобы удалить параметр, используйте функцию `delete_option()`. Единственный параметр для нее — имя удаляемого параметра:

```
<?php delete_option( 'prowp_display_mode' ); ?>
```

Хороший практичный подход — начинать все имена параметров с одного префикса, как `prowp_` в предыдущих примерах. Это полезно по двум причинам: уникальность и читабельность. Использование префикса поможет гарантировать уникальность имен параметров. Если у вас есть несколько параметров, прекрасная идея — хранить их в массиве (см. следующий раздел). Это также облегчает следование логике кода при наличии набора правил наименования, используемого для переменных, функций и т. д.

Параметры в WordPress не зарезервированы только для плагинов. Темы также могут создавать параметры для хранения своих специфических данных. Во многих темах, доступных в наше время, есть страница настроек, позволяющая индивидуализировать темы через настройки, а не через код.

Массив параметров

Каждый параметр, создаваемый вами в WordPress, добавляет новую запись в таблицу `wp_options` в базе данных. Поэтому отличная идея — хранить параметры в массиве, создавая меньше записей в базе данных и снижая число вызовов функции `update_option()`.

```
<?php
$prowp_options_arr = array(
    'prowp_display_mode'     =>  'Fright Night',
    'prowp_default_browser'  =>  'Chrome',
    'prowp_favorite_book'    =>  'Professional WordPress',
    );
update_option( 'prowp_plugin_options', $prowp_options_arr );
?>
```

С помощью данного кода вы создаете массив для хранения значений параметров плагина. Вместо того чтобы вызывать `update_option()` трижды и сохранять три записи в базе данных, вам нужно вызвать ее всего один раз и сохранить массив в параметре `prowp_plugin_options`. Это небольшой пример, но представьте себе коллекцию плагинов, хранящих 50 параметров в таблице `options` в базе данных. Таким образом можно в самом деле перегрузить базу данных и замедлить скорость

загрузки сайта повторяющимися запросами параметров для выборки или индивидуальной настройки.

Для возврата массива параметров используется та же функция `get_option()`, что и ранее:

```
<?php
$prowp_options_arr = get_option( 'prowp_plugin_options' );
$prowp_display_mode = $prowp_options_arr['prowp_display_mode'];
$prowp_default_browser = $prowp_options_arr['prowp_default_browser'];
$prowp_favorite_book = $prowp_options_arr['prowp_favorite_book'];
?>
```

В следующем разделе мы обсудим, как создавать меню для страницы настроек плагина.

Создание меню и подпунктов меню

WordPress поддерживает два разных способа создания произвольного меню для плагина. В первую очередь вам следует решить, где расположить страницу с настройками. Ссылка на страницу настроек может располагаться в собственном меню верхнего уровня (настройки плагина) или в подпункте в рамках существующего меню (Настройки ▸ Настройки плагина). В этом разделе изучаются оба варианта и их конфигурация.

Создание меню верхнего уровня

Первый метод, который мы рассмотрим, — это создание нового пункта меню верхнего уровня. Использование меню верхнего уровня имеет смысл, если у плагина есть несколько страниц настроек, которые следует разделить. Чтобы создать пункт меню верхнего уровня, используйте функцию `add_menu_page`, как показано ниже:

```
<?php add_menu_page( page_title, menu_title, capability,
menu_slug, function, icon_url, position ); ?>
```

Вот обзор допустимых параметров:

- `page_title` — текст, используемый для названия HTML (между тегами `<title>`).
- `menu_title` — текст, используемый как имя пункта меню в консоли.
- `capability` — минимальные права пользователя, позволяющие видеть меню.
- `menu_slug` — уникальное слаг-имя меню.
- `function` — отображает контент страницы настроек меню.
- `icon_url` — путь к произвольной иконке меню (по умолчанию `images/generic.png`).
- `position` — порядок отображения в меню. По умолчанию будет отображаться в конце структуры меню.

Вы также можете создать подпункты для нового пункта меню. Для этого используйте функцию `add_submenu_page()`:

```
add_submenu_page( parent, page_title, menu_title, capability,
menu_slug,[function] );
```

Создадим произвольное меню для плагина с несколькими подпунктами меню, как это показано на рис. 8.3.

```
<?php
// создаем произвольное меню для плагина
add_action( 'admin_menu', 'prowp_create_menu' );
function prowp_create_menu() {
    // создаем новое меню верхнего уровня
    add_menu_page( 'Halloween Plugin Page', 'Halloween Plugin',
        'manage_options', 'prowp_main_menu', 'prowp_main_plugin_page',
        plugins_url( '/images/wordpress.png', __FILE__ ) );
    // создаем подпункты меню: настройка и поддержка
    add_submenu_page( 'prowp_main_menu', 'Halloween Settings Page',
        'Settings', 'manage_options', 'halloween_settings',
        'prowp_settings_page' );
    add_submenu_page( 'prowp_main_menu', 'Halloween Support Page',
        'Support', 'manage_options', 'halloween_support', 'prowp_support_page' );
}
?>
```

Сначала вызываем зацепку-действие `admin_menu`. Эта зацепка запускается после создания базовой структуры меню панели администратора. После ее запуска вызываем произвольную функцию `prowp_create_menu()` для создания меню.

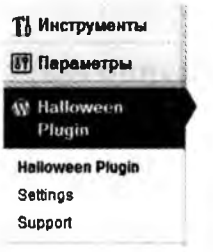


Рис. 8.3. Произвольный пункт меню верхнего уровня

Чтобы создать меню, вызываем функцию `add_menu_page()`. Первые два параметра задают названия страницы и меню. Устанавливаем уровень доступа для `manage_options`, чтобы меню мог видеть только администратор. Затем задаем слаг меню `propwp_main_menu`, то есть уникальный слаг меню. После этого следует произвольное название меню, в данном случае `prowp_main_plugin_page`. Не забывайте, что мы еще не создали функцию, поэтому при просмотре страницы настроек будем видеть предупреждение PHP. Наконец, задаем произвольную иконку для отображения логотипа WordPress.

Теперь пункт меню верхнего уровня создан и необходимо создать подпункты. В данном примере мы создаем два объекта подменю: Settings (Настройки) и Support (Поддержка). Чтобы сделать это, используем функцию `add_submenu_page()`.

Первый передаваемый параметр — слаг пункта меню верхнего уровня, к которому должны относиться подпункты. Помните, что он задан как `prowp_main_menu`, уникальный слаг меню плагина. Затем задаем названия страницы и меню, как перед этим. Так же задается уровень доступа для просмотра `manage_options`. Кроме этого, необходимо создать уникальный слаг меню для объектов подменю. В данном

примере произвольные имена имеют значения `halloween_settings` и `halloween_support`. Последним значением является произвольная функция для построения страницы настроек каждого объекта подменю.

Добавление в существующее меню

Далее мы изучим, как добавлять подпункты в уже существующее меню WordPress. У большинства плагинов только одна страница параметров, поэтому им не требуется отдельного пункта в меню верхнего уровня. Чтобы сделать это, нужно добавить страницу параметров плагина в существующее меню WordPress. Добавим подпункт в пункт меню Настройки:

```
<?php
add_action( 'admin_menu', 'prowp_create_settings_submenu' );
function prowp_create_settings_submenu() {
    add_options_page( 'Halloween Settings Page', 'Halloween Settings',
    'manage_options', 'halloween_settings_menu', 'prowp_settings_page' );
}
?>
```

WordPress поддерживает множество функций, делающих добавление подпунктов меню невероятно простым. Чтобы добавить подпункт меню Halloween Settings, используйте функцию `add_options_page()`. Первый параметр — название страницы, за которым следует отображаемое название подпункта. Как и в случае с другими вариантами меню, вы задаете уровень доступа к `manage_options`, чтобы меню могли видеть только администраторы. После этого задается уникальный идентификатор подпункта `halloween_settings_menu`. Затем вызывается произвольная функция `prowp_settings_page()` для построения страницы параметров. В предыдущем примере в конец меню настроек был добавлен произвольный подпункт Halloween Settings.

Ниже приведен список доступных в WordPress функций для подпунктов меню. Каждая функция может использоваться точно так же, как в предшествующем примере, — просто используйте имя одной из вызываемых функций:

- `add_dashboard_page()` — добавление подпункта меню в меню консоли.
- `add_posts_page()` — добавление подпункта меню в меню Записи.
- `add_media_page()` — добавление подпункта меню в меню Медиа.
- `add_links_page()` — добавление подпункта меню в меню Ссылки.
- `add_pages_page()` — добавление подпункта меню в меню Страницы.
- `add_comments_page()` — добавление подпункта меню в меню Комментарии.
- `add_plugins_page()` — добавление подпункта меню в меню Плагины.
- `add_theme_page()` — добавление подпункта меню в меню Внешний вид.
- `add_users_page()` — добавление подпункта меню в меню Пользователи.

- `add_management_page()` — добавление подпункта меню в меню Инструменты.
- `add_options_page()` — добавление подпункта меню в меню Настройки.

Теперь, когда вы создали пункты и подпункты меню, необходимо создать страницу параметров для отображения конфигурации плагина.

Создание страницы параметров

В WordPress 2.7 появился новый API настроек, который можно использовать для всех методов параметров, показанных в этом разделе. API настроек — это мощный набор функций, делающий сохранение параметров в WordPress простым и безопасным. Одним из основных преимуществ API настроек является то, что WordPress осуществляет проверки безопасности и вам не нужно включать временные значения в форму.

Первый метод создания страницы параметров, который мы будем осваивать, — это создание уникальной страницы параметров для меню верхнего уровня. Помните, что при использовании функций `add_menu_page()` и `add_submenu_page()` вы определяете функциональное имя пункта меню для отображения на странице параметров. Чтобы создать страницу параметров, нужно создать функцию для отображения параметров. Сначала создадим меню плагина:

```
<?php
// создаем произвольное меню для плагина
add_action( 'admin_menu', 'prowp_create_menu' );
function prowp_create_menu() {
    // создаем новое меню верхнего уровня
    add_menu_page( 'Halloween Plugin Page', 'Halloween Plugin',
        'manage_options', 'prowp_main_menu', 'prowp_settings_page',
        plugins_url( '/images/wordpress.png', __FILE__ ) );
    // вызываем функцию для регистрации настроек
    add_action( 'admin_init', 'prowp_register_settings' );
}
?>
```

Обратите внимание, что вы добавили новую зацепку-действие для `admin_init`, чтобы выполнить функцию `prowp_register_settings()`, как показано ниже:

```
<?php
function prowp_register_settings() {
    // регистрируем настройки
    register_setting( 'prowp-settings-group', 'prowp_options',
        'prowp_sanitize_options' );
}?>
```

Используя функцию API настроек `register_setting()`, вы определяете параметр, который собираетесь предложить на странице параметров плагина. У страницы настроек три параметра, но они будут храниться в едином массиве параметров, поэтому здесь нужно зарегистрировать только одну настройку. Первый параметр — имя группы параметров. Это обязательное поле должно быть групповым именем, идентифицирующим все параметры в наборе. Второй параметр — действительное

имя параметра, которое должно быть уникальным. Третий параметр — функция обратного вызова для очистки значений параметров. Теперь, когда три параметра зарегистрированы, нужно создать страницу параметров. Чтобы сделать это, создадим функцию `prowp_settings_page()` как вызываемую из меню:

```
<?php
function prowp_settings_page() {
?>
    <div class="wrap">
    <h2>Halloween Plugin Options</h2>
    <form method="post" action="options.php">
        <?php settings_fields( 'prowp-settings-group' ); ?>
        <?php $prowp_options = get_option( 'prowp_options' ); ?>
        <table class="form-table">
            <tr valign="top">
            <th scope="row">Name</th>
            <td><input type="text" name="prowp_options[option_name]"
                value="<?php echo esc_attr( $prowp_options['option_name'] ); ?>" />
            </td>
            </tr>
            <tr valign="top">
            <th scope="row">Email</th>
            <td><input type="text" name="prowp_options[option_email]"
                value="<?php echo esc_attr( $prowp_options['option_email'] ); ?>"
                 /></td>
            </tr>
            <tr valign="top">
            <th scope="row">URL</th>
            <td><input type="text" name="prowp_options[option_url]"
                value="<?php echo esc_url( $prowp_options['option_url'] ); ?>" />
                </td>
            </tr>
        </table>
        <p class="submit">
            <input type="submit" class="button-primary"
                value="Save Changes" />
        </p>
    </form>
    </div>
<?php
}
?>
```

Как видите, все это выглядит как стандартная форма с парой заметных отличий. Должен быть установлен тег `<form>`, чтобы публиковать в `options.php`. Внутри формы необходимо определить группу настроек, которую мы задали как `prowp-settings-group` при регистрации настроек. Это установит связь между параметрами и их значениями. Это делается с помощью следующей строки кода:

```
<?php settings_fields( 'prowp-settings-group' ); ?>
```

Теперь загрузим существующий массив параметров, если они есть, в переменную `$prowp_options` с помощью функции `get_option()`. Будем использовать эту переменную, чтобы отображать существующие параметры, заданные в форме.

Затем построим таблицу для отображения параметров формы. Обратите внимание на имя поля формы, которое должно быть в формате `option_name[field_name]`, поскольку вы храните все параметры в едином массиве.

```
<input type="text" name="prowp_options[option_email]"
    value="<?php echo esc_attr( $prowp_options['option_email'] ); ?>" />
```

После того как вы отобразили все поля формы, необходимо отобразить и кнопку Отправить (Submit) для отправки формы и сохранения параметров. Последний шаг — создание функции `prowp_sanitize_options()`. Она будет использоваться для очистки всех данных, передаваемых настройкам плагина, перед сохранением в базе данных. Данный шаг исключительно важен, поскольку неочищенные данные потенциально способны вскрыть уязвимость в плагине.

```
<?php
function prowp_sanitize_options( $input ) {
    $input['option_name'] = sanitize_text_field( $input['option_name'] );
    $input['option_email'] = sanitize_email( $input['option_email'] );
    $input['option_url'] = esc_url( $input['option_url'] );
    return $input;
}
?>
```

Обратите внимание, как каждое значение параметра очищается с помощью особой функции. Имя параметра использует функцию WordPress `sanitize_text_field()`, чтобы убрать любые теги HTML, XML и PHP из передаваемых значений. Функция WordPress `sanitize_email()` используется для очистки значений электронного адреса, а `esc_url()` — для очистки значения URL.

Готово! Вы только что создали простейшую страницу параметров плагина, используя API настроек WordPress. В листинге 8.1 приведен код для построения страницы параметров целиком.

Листинг 8.1. Построение страницы параметров (prowp2-settings-api-plugin.zip)

```
<?php
// создаем произвольное меню для плагина
add_action( 'admin_menu', 'prowp_create_menu' );
function prowp_create_menu() {

    // создаем новое меню верхнего уровня
    add_menu_page( 'Halloween Plugin Page', 'Halloween Plugin',
        'manage_options', 'prowp_main_menu', 'prowp_settings_page',
         plugins_url( '/images/wordpress.png', __FILE__ ) );
    // вызываем функцию для регистрации настроек
    add_action( 'admin_init', 'prowp_register_settings' );

}
function prowp_register_settings() {

    // регистрируем настройки
    register_setting( 'prowp-settings-group',
```

```
        'prowp_options', 'prowp_sanitize_options' );

}
function prowp_sanitize_options( $input ) {
    $input['option_name'] =
        sanitize_text_field( $input['option_name'] );
    $input['option_email'] = sanitize_email( $input['option_email'] );
    $input['option_url'] = esc_url( $input['option_url'] );
    return $input;
}
function prowp_settings_page() {
?>
    <div class="wrap">
    <h2>Halloween Plugin Options</h2>
    <form method="post" action="options.php">
        <?php settings_fields( 'prowp-settings-group' ); ?>
        <?php $prowp_options = get_option( 'prowp_options' ); ?>
        <table class="form-table">
            <tr valign="top">
            <th scope="row">Name</th>
            <td><input type="text" name="prowp_options[option_name]"
                value="<?php echo esc_attr( $prowp_options['option_name'] );?>
                " /></td>
            </tr>
            <tr valign="top">
            <th scope="row">Email</th>
            <td><input type="text" name="prowp_options[option_email]"
                 value="<?php echo esc_attr( $prowp_options['option_email'] ); ?>
                " /></td>
            </tr>
            <tr valign="top">
            <th scope="row">URL</th>
            <td><input type="text" name="prowp_options[option_url]"
                value="<?php echo esc_url( $prowp_options['option_url'] ); ?>" />
            </td>
            </tr>
        </table>
        <p class="submit">
            <input type="submit" class="button-primary"    value="Save Changes" />
        </p>
    </form>
    </div>
<?php
}
?>
```

Второй метод создания страницы параметров — добавление настроек плагина в существующую страницу настроек в WordPress, как показано на рис. 8.4. Понадобится также использовать API настроек WordPress, чтобы осуществлять перехват на этих страницах и добавлять настройки плагина.

Рассмотрим код для создания собственного раздела произвольных настроек. В приведенном далее примере мы будем добавлять новый раздел настроек в конец страницы **Настройки ▸ Чтение**. Он будет содержать параметры для плагина.

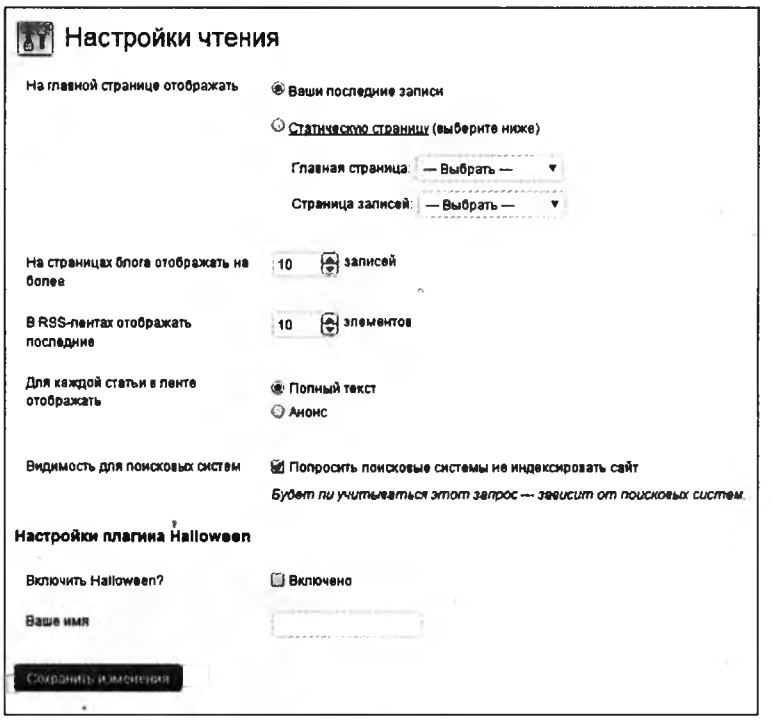


Рис. 8.4. Установка произвольных настроек

```
<?php
// выполняем функцию раздела настроек
add_action( 'admin_init', 'prowp_settings_init' );
function prowp_settings_init() {
    // создаем новый раздел настроек в Параметры > Чтение
    add_settings_section( 'prowp_setting_section', 'Halloween Plugin Settings',
        'prowp_setting_section', 'reading' );
    // регистрируем индивидуальные настройки
    add_settings_field( 'prowp_setting_enable_id', 'Enable Halloween Feature?',
        'prowp_setting_enabled', 'reading', 'prowp_setting_section' );
    add_settings_field( 'prowp_saved_setting_name_id', 'Your Name',
        'prowp_setting_name', 'reading', 'prowp_setting_section' );
    // регистрируем настройки с помощью массива значений
    register_setting( 'reading', 'prowp_setting_values' );
}
?>
```

Сначала используем зацепку-действие `admin_init` для загрузки произвольной функции `prowp_settings_init()` перед генерированием страницы администратора. Затем вызовем функцию `add_settings_section()` для создания нового раздела:

```
<?php
add_settings_section( 'prowp_setting_section', 'Halloween Plugin Settings',
    'prowp_setting_section', 'reading' );
?>
```

Первый передаваемый параметр — уникальный ID этого раздела. Второй параметр — имя на выходе, отображаемое на странице. После этого передается имя функции обратного вызова для отображения самого раздела. Последний параметр задает страницу настроек для добавления в раздел. Значения WordPress, принятые по умолчанию: `general`, `writing`, `reading`, `discussion`, `media`, `privacy` и `permalink`.

```
<?php
    // регистрируем индивидуальные настройки
    add_settings_field( 'prowp_setting_enable_id', 'Enable Halloween Feature?',
        'prowp_setting_enabled', 'reading', 'prowp_setting_section' );
    add_settings_field( 'prowp_saved_setting_name_id', 'Your Name',
        'prowp_setting_name', 'reading', 'prowp_setting_section' );
?>
```

Теперь, зарегистрировав раздел произвольных настроек, следует зарегистрировать параметры индивидуальных настроек. Чтобы сделать это, используем функцию `add_settings_field()`. Первый передаваемый ей параметр — уникальный ID поля. Затем передается название поля, отображающееся слева от поля параметра. Третий параметр — имя функции обратного вызова, используемой для отображения поля параметра. Четвертый параметр — страница настроек, на которой должно отображаться поле. Последний параметр — имя раздела, в который добавляется поле, в данном примере — `prowp_setting_section`, созданный посредством вызова функции `add_setting_section()`.

```
<?php
register_setting( 'reading', 'prowp_setting_values', 'prowp_sanitize_settings' );
?>
```

Теперь необходимо зарегистрировать поле настройки. В данном примере мы зарегистрируем две разные настройки: одну для активации/деактивации кнопки-флажка и одну для имени пользователя. Несмотря на то что полей настроек два, значения обоих будут храниться в одном массиве, поэтому нужно зарегистрировать только одну настройку под названием `prowp_setting_values`. Первый передаваемый параметр — группа параметров. В данном примере мы будем сохранять параметры в группе чтения с остальными параметрами чтения. Второй параметр — имя параметра. Имя параметра должно быть уникальным. Оно используется для возврата значения параметра. Третий необязательный параметр может быть установлен для произвольной функции, используемой для очистки значений параметра. В данном примере создается функция `prowp_sanitize_settings()` для очистки значений параметра, вводимых пользователем.

```
<?php
function prowp_sanitize_settings( $input ) {
    $input['enabled'] = ( $input['enabled'] == 'on' ) ? 'on' : '';
    $input['name'] = sanitize_text_field( $input['name'] );
    return $input;
}
?>
```

Как всегда, нам потребуется очищать все значения параметра, вводимые пользователем. Параметр активируется посредством кнопки-флажка и поэтому может принимать одно из двух значений: отмечен или нет. В предыдущем примере для определения значения «Enabled» («включено») используется трехзначный оператор PHP. Если кнопка-флажок соответствует параметру «включено», вы знаете, что значение активировано и значение параметра сохраняется как «включено». Если нет — параметр сохранит значение как пустое, это означает, что кнопка-флажок не активирована. Теперь, после регистрации раздела настроек, нужно создать произвольные функции для его отображения. Первой функцией будет `prowp_setting_section()`, которую мы вызвали при создании раздела настроек:

```
<?php
function prowp_setting_section() {
  . echo '<p>Configure the Halloween plugin options below</p>';
}
?>
```

Здесь можно задать подзаголовок для раздела настроек. Этот раздел отлично подходит для инструкций к плагинам, конфигурационной информации и т. д. Затем необходимо создать функцию для отображения первого поля настроек:

```
<?php
function prowp_setting_enabled() {
    // получаем настройки плагина
    $prowp_options = get_option( 'prowp_setting_values' );
    // отображаем форму с чекбоксами
    echo '<input '.checked( $prowp_options['enabled'], 'on', false ).'
        name="prowp_setting_values[enabled]" type="checkbox" /> Enabled';
}
?>
```

Это функция обратного вызова, которую вы определили при использовании функции `add_settings_field()`. Первый шаг — загрузка массива параметров, если таковой существует. Поскольку этот параметр представлен кнопкой-флажком, ясно, что в случае его установки кнопка должна быть активирована. В данном примере используется функция WordPress `checked()`. У нее есть три параметра. Первый и второй — два значения для сравнения. Если они совпадают, функция отобразит `checked="checked"`, активируя таким образом элемент формы. Третий параметр определяет, отображать значение или просто возвращать его. В данном случае нам нужно просто вернуть его, поэтому данному параметру присваивается значение `false`.

Теперь отобразим поле актуальных настроек, которое будет использоваться в разделе настроек. Имя поля для ввода должно совпадать с именем настройки, зарегистрированным ранее. Поскольку параметры хранятся как массив, необходимо определить значение имени массива. В данном примере это `prowp_setting_values[enabled]`. Таким образом API настроек поймет, какой параметр сохранять и где именно. Поле активации кнопки-флажка будет отображаться в нижней части страницы **Настройки ▸ Чтение**. Теперь необходимо создать функцию для поля второй настройки:

```
<?php
function prowp_setting_name() {
    // получаем значение настройки
    $prowp_options = get_option( 'prowp_setting_values' );
    // отображаем текстовое поле
    echo '<input type="text" name="prowp_setting_values[name]"
        value="'.esc_attr( $prowp_options['name'] ).'" />';
}
?>
```

Как и в случае с кнопкой-флажком в первую очередь загружаем текущее значение параметра. Затем отображаем поле ввода текста с тем же именем, которое было определено ранее в функции `register_setting()`. Как всегда следует убедиться, что значения прошли процедуру обработки до отображения в поле формы.

Готово! Вы успешно создали раздел произвольных настроек и добавили его во вкладку Настройки ▸ Чтение. В листинге 8.2 приведен полный код.

Листинг 8.2. Раздел произвольных настроек (prowp2-reading-settings-plugin.zip)

```
<?php
// выполняем функцию раздела настроек
add_action( 'admin_init', 'prowp_settings_init' );
function prowp_settings_init() {
    // создаем новый раздел настроек в Параметры > Чтение
    add_settings_section( 'prowp_setting_section',
        'Halloween Plugin Settings', 'prowp_setting_section',
        'reading' );
    // регистрируем индивидуальные настройки
    add_settings_field( 'prowp_setting_enable_id', 'Enable Halloween Feature?',
    'prowp_setting_enabled', 'reading', 'prowp_setting_section' );
    add_settings_field( 'prowp_saved_setting_name_id', 'Your Name',
        'prowp_setting_name', 'reading', 'prowp_setting_section' );
    // регистрируем настройки с помощью массива значений
    register_setting( 'reading', 'prowp_setting_values',
        'prowp_sanitize_settings' );
}
function prowp_sanitize_settings( $input ) {
    $input['enabled'] = ( $input['enabled'] == 'on' ) ? 'on' : '';
    $input['name'] = sanitize_text_field( $input['name'] );
    return $input;
}
// раздел настроек
function prowp_setting_section() {
    echo '<p>Configure the Halloween plugin options below</p>';
}
// создаем чекбокс
function prowp_setting_enabled() {
    // получаем настройки плагина
    $prowp_options = get_option( 'prowp_setting_values' );
    // отображаем чекбокс
    echo '<input '.checked( $prowp_options['enabled'], 'on', false )
        .' name="prowp_setting_values[enabled]" type="checkbox" />
         Enabled';
}
```

```
// создаем текстовое поле для названия
function prowp_setting_name() {
    //load the option value
    $prowp_options = get_option( 'prowp_setting_values' );
    //display the text form field
    echo '<input type="text" name="prowp_setting_values[name]"
        value="'.esc_attr( $prowp_options['name'] ).'" />';
}
?>
```

Интеграция с WordPress

Интеграция плагина с WordPress — значимый шаг для взаимодействия пользователей с плагином через консоль администратора. WordPress поддерживает множество различных областей, куда можно интегрировать плагин, включая метаполя, боковую панель, виджеты консоли и произвольные сокращенные коды.

Создание метаполя

У WordPress есть множество метаполей на экранах добавления новой записи или страницы. Эти метаполя используются для добавления дополнительной информации к записям, страницам и контенту.

Метаполя можно создать в плагине, используя функцию WordPress `add_meta_box()`. Она принимает семь параметров:

```
<?php add_meta_box( $id, $title, $callback, $page, $context, $priority, $callback_
args ); ?>
```

Каждый параметр помогает определить, где и как отображается метаполе.

- `$id` (атрибут ID CSS для метаполя);
- `$title` (название страницы, отображаемое в заголовке метаполя);
- `$callback` (имя произвольной функции для отображения в информации метаполя);
- `$page` (страница, на которой должно отображаться метаполе, — `'post'`, `'page'` или название индивидуального типа записи);
- `$context` (часть страницы, в которой должно быть отображено метаполе, — `'normal'`, `'advanced'` или `'side'`);
- `$priority` (где именно в выбранной части страницы должно отображаться метаполе — `'high'`, `'core'`, `'default'` или `'low'`);
- `$callback_args` (аргументы для передачи функции обратного вызова).

Теперь, понимая функцию `add_meta_box()`, можно создать первое произвольное метаполе в WordPress:

```
<?php
add_action( 'add_meta_boxes', 'prowp_meta_box_init' );
// функции для добавления метаполя и сохранения данных
function prowp_meta_box_init() {
    // создаем произвольное метаполе
    add_meta_box( 'prowp-meta', 'Product Information',
        'prowp_meta_box', 'post', 'side', 'default' );
}
?>
```

Первый шаг для добавления собственного метаполя — использование зацепки-действия `add_meta_boxes` для выполнения произвольной функции `prowp_meta_box_init()`. В этой функции вы вызываете функцию `add_meta_box()` для создания произвольного метаполя для информации о товаре.

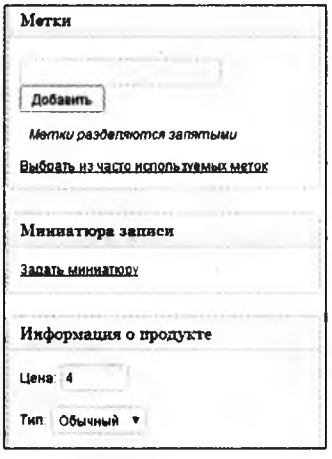


Рис. 8.5. Произвольное метаполе

Задаем атрибут ID CSS `prowp-meta` для метаполя. Второй параметр — название, определяемое для информации о товаре. Следующий параметр — произвольная функция `prowp_meta_box()`, отображающая HTML для метаполя. Затем определяется отображение метаполя на странице записи и на боковой панели. Наконец, выставляем положение `default`. Теперь создаем произвольную функцию `prowp_meta_box()` для отображения полей метаполя:

```
function prowp_meta_box( $post, $box ) {

    // извлекаем значения произвольного метаполя
    $prowp_featured = get_post_meta( $post->ID, '_prowp_type', true );
    $prowp_price = get_post_meta( $post->ID, '_prowp_price', true );
    // временные значения из соображений безопасности
    wp_nonce_field( plugin_basename( __FILE__ ), 'prowp_save_meta_box' );

    // форма метаполя
    echo '<p>Price: <input type="text" name="prowp_price"
        value="'.esc_attr( $prowp_price ).'" size="5" /></p>';
```

```
    echo '<p>Type:
        <select name="prowp_product_type" id="prowp_product_type">
            <option value="0" '
                .selected( $prowp_featured, 'normal', false ). '>Normal
            </option>
            <option value="special" '
                .selected( $prowp_featured, 'special', false ). '>Special
            </option>
            <option value="featured" '
                .selected( $prowp_featured, 'featured', false ). '>Featured
            </option>
            <option value="clearance" '
                .selected( $prowp_featured, 'clearance', false ). '>Clearance
            </option>
        </select></p>';

}
```

Первый шаг в рамках произвольной функции — получение сохраненных значений для метаполя. Если вы создаете новую запись, сохраненных значений еще не существует. После этого отобразим элементы формы в метаполе. Обратите внимание, что вам не нужны теги `<form>` или кнопка подтверждения отправки. Также обратите внимание, что вы используете функцию `wp_nonce_field()` для создания произвольного поля временного значения в форме.

Только что созданная произвольная функция сгенерирует произвольное метаполе, как показано на рис. 8.5.

Теперь, получив метаполе и элементы формы, нужно сохранить эти данные при сохранении записи. Чтобы сделать это, создадим произвольную функцию `prowp_save_meta_box()`, запускающуюся зацепкой-действием `save_post`:

```
<?php
// сохраняем данные метаполя во время сохранения записи
add_action( 'save_post', 'prowp_save_meta_box' );
function prowp_save_meta_box( $post_id ) {

    // обрабатываем данные формы, если установлена переменная $_POST
    if( isset( $_POST['prowp_product_type'] ) ) {
        // если включено автосохранение, пропускаем этап сохранения данных метаполя
        if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE )
            return;
        // проверяем временное значение из соображений безопасности
        check_admin_referer( plugin_basename( __FILE__ ), 'prowp_save_meta_box' );

     // сохраняем данные метаполя в произвольных полях записи, используя префикс ID
        update_post_meta( $post_id, '_prowp_type',
            sanitize_text_field( $_POST['prowp_product_type'] ) );
        update_post_meta( $post_id, '_prowp_price',
            sanitize_text_field ( $_POST['prowp_price'] ) );

    }

}
?>
```

Зацепка-действие `save_post` запускается при сохранении записи в WordPress. Поскольку мы хотим всего лишь работать с произвольными метаданными в метаполе, сначала нужно убедиться, что задано значение `$_POST['prowp_product_type']`. Затем нужно убедиться, что сохраняемая запись — активная, а не версия автосохранения. Чтобы сделать это, проверьте, не сохраняется ли запись автоматически, и если сохраняется, отключите эту функцию. Следующий шаг — проверка того, что временное значение является ожидаемым значением. Если запись активна, а элементы формы заданы, вы сохраняете данные формы. После проведения всех проверок используем `update_post_meta()` для сохранения данных метаполя для записи.

Как видите, в качестве первого параметра `update_post_meta()` посылается ID. Таким образом WordPress сообщается, какие метаданные записи будут прикреплены. Затем передается имя обновляемого метаключа. Обратите внимание, что имя метаключа содержит префикс с нижним подчеркиванием. Это предохраняет данные значения от появления в списке произвольных полей метаполя на экране редактирования записи. Поскольку мы предоставили UI для редактирования этих значений, они не нужны в произвольных полях метаполя. Последний посылаемый параметр — новое значение для метаключа, очищенное функцией WordPress `sanitize_text_field()`.

Теперь у вас есть полностью функциональное произвольное метаполе, сохраняющее индивидуальные данные для каждой записи. В листинге 8.3 приведен полный код произвольного метаполя.

Листинг 8.3. Произвольное метаполе (prowp2-custom-meta-box.zip)

```
<?php
add_action( 'add_meta_boxes', 'prowp_meta_box_init' );
// функции для добавления и сохранения данных метаполя
function prowp_meta_box_init() {

    // создаем произвольное метаполе
    add_meta_box( 'prowp-meta', 'Product Information',
        'prowp_meta_box', 'post', 'side', 'default' );

}
function prowp_meta_box( $post, $box ) {

    // извлекаем значения произвольного метаполя
    $prowp_featured = get_post_meta( $post->ID, '_prowp_type', true );
    $prowp_price = get_post_meta( $post->ID, '_prowp_price', true );
    // временное значение из соображений безопасности
    wp_nonce_field( plugin_basename( __FILE__ ), 'prowp_save_meta_box' );

    // элементы формы метаполя
    echo '<p>Price: <input type="text" name="prowp_price"
        value="'.esc_attr( $prowp_price ).'" size="5" /></p>';
    echo '<p>Type:
        <select name="prowp_product_type" id="prowp_product_type">
            <option value="0" '
                .selected( $prowp_featured, 'normal', false )
                . '>Normal</option>
```

```
            <option value="special" '
                .selected( $prowp_featured, 'special', false )
                    . '>Special</option>
            <option value="featured" '
                .selected( $prowp_featured, 'featured', false )
                    . '>Featured</option>
            <option value="clearance" '
                .selected( $prowp_featured, 'clearance', false )
                    . '>Clearance</option>
        </select></p>';

}
// сохраняем данные метаполя при сохранении записи
add_action( 'save_post', 'prowp_save_meta_box' );
function prowp_save_meta_box( $post_id ) {

    // обрабатываем данные формы, если установлена переменная $_POST
    if( isset( $_POST['prowp_product_type'] ) ) {
        // если включено автосохранение, пропускаем сохранение данных метаполя
        if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE )
            return;
        // проверяем временное значение из соображений безопасности
        check_admin_referer(
            plugin_basename( __FILE__ ), 'prowp_save_meta_box' );

     // сохраняем данные метаполя в произвольных полях записи, используя префикс ID
        update_post_meta( $post_id, '_prowp_type',
            sanitize_text_field( $_POST['prowp_product_type'] ) );
        update_post_meta( $post_id, '_prowp_price',
            sanitize_text_field( $_POST['prowp_price'] ) );

    }

}
?>
```

Теперь, сохранив поле метаданных, вы, пожалуй, захотите отобразить их где-нибудь. Сохраненные метаданные легко отображать в теме, применяя функцию **get_post_meta** внутри Loop:

```
<?php
    $prowp_type = get_post_meta( $post->ID, '_prowp_type', true );
    $prowp_price = get_post_meta( $post->ID, '_prowp_price', true );
    echo '<p>Price: ' .esc_html( $prowp_price ). '</p>';
    echo '<p>Type: ' .esc_html( $prowp_type ). '</p>';
?>
```

Добавление произвольного метаполя — отличный способ расширения данных записей и страниц, интуитивно понятный пользователям.

Сокращенные коды

WordPress снабжен API сокращенных кодов, который может использоваться для создания функциональности сокращенного кода в плагинах. Сокращенные коды,

по сути, являются макрокодами, которые можно встроить в запись, страницу или запись индивидуального типа. При отображении эти сокращенные коды заменяются каким-либо другим типом контента. Рассмотрим простой пример использования API сокращенного кода:

```
<?php
add_shortcode( 'mytwitter', 'prowp_twitter' );
function prowp_twitter() {
    return '<a href="http://twitter.com/williamsba">@williamsba</a>';
}
?>
```

Теперь всякий раз при использовании в контенте сокращенного кода `[mytwitter]` он будет заменяться ссылкой HTML на учетную запись в Twitter при отображении в браузере. Как видите, это мощная возможность WordPress, из которой можно извлечь пользу для множества плагинов, в которые частенько помещаются небольшие фрагменты JavaScript для добавления кнопки или рекламы в том или ином месте записи.

Сокращенные коды также можно сконфигурировать для приема атрибутов. Это весьма полезно для передачи аргументов произвольным функциям, изменяющим выходные данные сокращенного кода на основе этих аргументов. Изменим функцию сокращенного кода, чтобы она принимала параметр сайта:

```
<?php
add_shortcode( 'mytwitter', 'prowp_twitter' );
function prowp_twitter( $atts, $content = null ) {
    extract( shortcode_atts( array(
        'person' => 'brad' // set attribute default
    ), $atts ) );
    if ( $person == 'brad' ) {
        return '<a href="http://twitter.com/williamsba">@williamsba</a>';
    }elseif ( $person == 'david' ) {
        return '<a href="http://twitter.com/mirmillo">@mirmillo</a>';
    }elseif ( $person == 'hal' ) {
        return '<a href="http://twitter.com/freeholdhal">@freeholdhal</a>';
    }
}
?>
```

Этот код создает тот же сокращенный код, что и раньше, но теперь для него определяется атрибут под названием `person`. С его помощью вы можете определить, для кого именно будет отображаться ссылка на Twitter. Для отображения URL Twitter для Дэвида нужно использовать сокращенный код `[mytwitter person="david"]`. Или же можно легко отобразить URL Twitter URL для Хэла: `[mytwitter person="hal"]`. Сокращенные коды также могут принимать несколько атрибутов в виде массива, заданного в функции сокращенного кода.

Создание виджета

Виджеты — распространенные возможности, входящие во многие плагины WordPress. Создавая виджет посредством плагина, вы можете дать пользователю

возможность легко добавить информацию плагина на боковую панель или в другие пригодные для виджета области.

Чтобы понять, как работает виджет, полезно обратиться к обзору класса `WP_Widget` в WordPress. Класс виджетов поддерживает встроенную функцию для встраивания виджета, каждого — для своей цели, как показывает следующий код:

```
<?php
class My_Widget extends WP_Widget {
    function My_Widget() {
        // код виджета
    }
    function form($instance) {
        // форма виджета в Консоли
    }
    function update($new_instance, $old_instance) {
        // сохраняем значения виджета
    }
    function widget($args, $instance) {
        // отображаем виджет
    }
}
?>
```

В качестве примера создадим базовый виджет Bio. Он позволит задавать имя человека и произвольную информацию о нем для отображения на поддерживающей виджеты боковой панели в WordPress.

Первый шаг в создании виджета — использование подходящей зацепки для его инициализации. Такая зацепка называется `widgets_init` и активируется сразу после регистрации предустановленных виджетов WordPress:

```
add_action( 'widgets_init', 'prowp_register_widgets' );
function prowp_register_widgets() {
    register_widget( 'prowp_widget' );
}
```

Вызов зацепки-действия `widgets_init` запускает функцию `prowp_register_widgets()`, как показано в предшествующем коде. Здесь мы регистрируем виджет с названием `pro_widget`. При необходимости посредством данной функции можно регистрировать несколько виджетов.

Улучшенный API виджетов, появившийся в WordPress 2.8, делает создание виджета гораздо более простым, чем раньше. Для начала необходимо расширить предустановленный класс `WP_Widget`, создав новый класс с уникальным именем:

```
class prowp_widget extends WP_Widget {
```

Теперь добавим первую функцию, имя которой должно совпадать с уникальным именем класса. Функция такого типа называется *конструктор* (*constructor*):

```
function prowp_widget() {
    $widget_ops = array(
```

```
        'classname'   => 'prowp_widget_class',
        'description' => 'Example widget that displays a user\'s bio.' );
    $this->WP_Widget( 'prowp_widget', 'Bio Widget', $widget_ops );
}
```

В функции `prowp_widget()` определяем имя класса для виджета. Имя класса — это класс CSS, который будет добавлен к тегу HTML, включающему в себя виджет при его отображении. В зависимости от темы класс CSS может оказаться в `<div>`, `<aside>`, `<li>` или каком-либо еще HTML-теге. Вы также задаете описание виджета. Оно отображается на консоли виджета ниже имени виджета. Затем данные параметры передаются `WP_Widget`. Вы также передаете имя ID CSS (`prowp_widget_class`) и имя виджета (Bio Widget).

Теперь создадим функцию для встраивания формы настроек виджета. Настройки располагаются на администраторской странице виджета, раскрываясь для каждого виджета, перечисленного на боковой панели. Класс виджета делает процесс крайне простым, как показывает код ниже:

```
function form( $instance ) {
    $defaults = array(
        'title' => 'My Bio',
        'name'  => 'Michael Myers',
        'bio'   => '' );
    $instance = wp_parse_args( (array) $instance, $defaults );
    $title = $instance['title'];
    $name = $instance['name'];
    $bio = $instance['bio'];
    ?>
        <p>Title:
            <input class="widefat"
                name="<?php echo $this->get_field_name( 'title' ); ?>"
                type="text" value="<?php echo esc_attr( $title ); ?>" /></p>
        <p>Name:
            <input class="widefat"
                name="<?php echo $this->get_field_name( 'name' ); ?>"
                type="text" value="<?php echo esc_attr( $name ); ?>" /></p>
        <p>Bio:
            <textarea class="widefat"
                name="<?php echo $this->get_field_name( 'bio' ); ?>" >
                <?php echo esc_textarea( $bio ); ?></textarea></p>
    <?php
}
```

Первое, что нужно сделать, — это определить значения виджета по умолчанию. На тот случай, если пользователь не заполняет эти настройки, вы можете приписать им любые предустановленные значения. В данном примере мы задаем название по умолчанию My Bio, а имя по умолчанию Майкл Майерс (Michael Myers). Теперь задействуем значения объекта, то есть настройки виджета. Если виджет был только что добавлен на боковую панель, никаких настроек еще не сохранено, поэтому значения будут пустыми. Наконец, отображаем три поля формы для настроек виджета: название, имя и биография. Первые два параметра используют поля для ввода текста, а биография (bio) — поле области текста. Обратите внимание, что вам не

нужны теги <form> или кнопка подтверждения; это сделает для вас класс виджета. Не забывайте использовать надлежащие функции обработки при отображении данных, в нашем случае esc_attr() для двух текстовых полей и esc_textarea() для поля области текста. Затем нужно сохранить настройки виджета, используя функцию класса виджета update():

```
function update( $new_instance, $old_instance ) {
    $instance = $old_instance;
    $instance['title'] = sanitize_text_field( $new_instance['title'] );
    $instance['name'] = sanitize_text_field( $new_instance['name'] );
    $instance['bio'] = sanitize_text_field( $new_instance['bio'] );
    return $instance;
}
```

Эта функция вполне ясная. Вы видите, что не нужно сохранять настройки, класс виджета сделает это сам. Передаем значения $new_instance для каждого из полей настроек. Также используем sanitize_text_field() для очищения любого HTML-кода, который мог быть введен. Если вы хотите принять значения HTML, следует использовать wp_kses(), о чем говорилось ранее, в разделе «Валидация и очистка данных» этой главы.

Последняя функция в классе prowp_widget занимается отображением виджета:

```
function widget( $args, $instance ) {
    extract( $args );
    echo $before_widget;
    $title = apply_filters( 'widget_title', $instance['title'] );
    $name = ( empty( $instance['name'] ) ) ? ' ' : $instance['name'];
    $bio = ( empty( $instance['bio'] ) ) ? ' ' : $instance['bio'];
    if ( !empty( $title ) ) { echo $before_title . esc_html( $title )
        . $after_title; };
    echo '<p>Name: ' . esc_html( $name ) . '</p>';
    echo '<p>Bio: ' . esc_html( $bio ) . '</p>';
    echo $after_widget;
}
```

Первое, что мы делаем, — извлекаем параметр $args. Эта переменная хранит некоторые глобальные значения темы, такие как $before_widget и $after_widget. Эти переменные могут использоваться разработчиками темы для определения того, какой код будет обрамлять виджет, например произвольный тег . После извлечения параметра $args отображаем переменную $before_widget. $before_title и $after_title также задаются в этой переменной. Это полезно для передачи произвольных тегов HTML для включения между ними названия виджета.

Теперь отобразим значения виджета. Название показывается первым и помещается между $before_title и $after_title. Затем покажем имя и биографию. Не забывайте применять исключение для значений виджета по соображениям безопасности. Наконец, отобразим значение $after_widget.

Готово! Вы только что создали пользовательский виджет для плагина, используя класс виджета в WordPress. Не забывайте, что, используя новый класс виджета,

вы можете добавить множество копий одного и того же виджета на боковую панель или дополнительные боковые панели. В листинге 8.4 приведен полный код виджета.

Листинг 8.4. Пользовательский виджет (prowp2-custom-widget.zip)

```
<?php
// используем зацепку widgets_init, чтобы запустить произвольную функцию
add_action( 'widgets_init', 'prowp_register_widgets' );
// регистрируем виджет
function prowp_register_widgets() {
    register_widget( 'prowp_widget' );

}
//prowpwidget class
class prowp_widget extends WP_Widget {
    // код виджета
    function prowp_widget() {

        $widget_ops = array(
            'classname'   => 'prowp_widget_class',
            'description' => 'Example widget that
                displays a user\'s bio.' );
        $this->WP_Widget( 'prowp_widget', 'Bio Widget',
             $widget_ops );

    }
     // создаем форму настроек виджета
    function form( $instance ) {
        $defaults = array(
            'title' => 'My Bio',
            'name'  => 'Michael Myers',
            'bio'   => '' );
        $instance = wp_parse_args( (array) $instance, $defaults );
        $title = $instance['title'];
        $name = $instance['name'];
        $bio = $instance['bio'];
        ?>
            <p>Title:
                <input class="widefat" name="<?php
                echo $this->get_field_name( 'title' );
                ?>" type="text"
                    value="<?php echo esc_attr( $title ); ?>" /></p>
            <p>Name:
                <input class="widefat" name="<?php
                echo $this->get_field_name( 'name' );
                ?>" type="text"
                    value="<?php echo esc_attr( $name ); ?>" /></p>
            <p>Bio:
                <textarea class="widefat"
                name="<?php echo $this->get_field_name( 'bio' ); ?>"
                ><?php echo esc_textarea( $bio ); ?></textarea></p>
        <?php
    }
    // сохраняем настройки виджета
```

```
    function update( $new_instance, $old_instance ) {

        $instance = $old_instance;
        $instance['title'] =
            sanitize_text_field( $new_instance['title'] );
        $instance['name'] =
            sanitize_text_field( $new_instance['name'] );
        $instance['bio'] =
            sanitize_text_field( $new_instance['bio'] );
        return $instance;

    }
    // отображаем виджет
    function widget( $args, $instance ) {
        extract( $args );
        echo $before_widget;
        $title = apply_filters( 'widget_title', $instance['title'] );
        $name = ( empty( $instance['name'] ) )
            ? ' ' : $instance['name'];
        $bio = ( empty( $instance['bio'] ) )
            ? ' ' : $instance['bio'];
        if ( !empty( $title ) ) { echo $before_title
            . esc_html( $title ) . $after_title; };
        echo '<p>Name: ' . esc_html( $name ) . '</p>';
        echo '<p>Bio: ' . esc_html( $bio ) . '</p>';

        echo $after_widget;

    }
}
?>
```

Создание консольного виджета

В WordPress 2.7 появились консольные виджеты (Dashboard Widgets), представляющие собой виджеты, отображаемые в основной консоли вашей копии WordPress. Вместе с этими новыми виджетами идет API консольных виджетов для удобной работы с этими виджетами.

Чтобы создать произвольный консольный виджет, используем функцию `wp_add_dashboard_widget()`:

```
<?php
add_action( 'wp_dashboard_setup', 'prowp_add_dashboard_widget' );
// вызываем функцию для создания консольного виджета
function prowp_add_dashboard_widget() {
    wp_add_dashboard_widget( 'prowp_dashboard_widget',
        'Pro WP Dashboard Widget', 'prowp_create_dashboard_widget' );
}
// функция для отображения содержания консольного виджета
function prowp_create_dashboard_widget() {
    echo '<p>Hello World! This is my Dashboard Widget</p>';
}
?>
```

Сначала вызываем зацепку-действие `wp_dashboard_setup` для выполнения функции построения произвольного консольного виджета. Эта зацепка активируется после построения всех консольных виджетов по умолчанию. Теперь выполним функцию `wp_add_dashboard_widget()` для создания консольного виджета. Первый параметр — слаг ID виджета. Он используется для имени класса CSS и ключа в массиве виджетов. Следующий параметр — имя виджета. Последний передаваемый параметр — имя произвольной функции для отображения контента виджета. В качестве необязательного четвертого параметра может быть послана контрольная функция обратного вызова. Она будет использоваться для обработки любых элементов формы, которые могут существовать в консольном виджете.

После выполнения функции `wp_add_dashboard_widget()` вызывается произвольная функция для отображения контента виджета. В данном примере отображается простая строка. Результат — пользовательский консольный виджет, показанный на рис. 8.6.

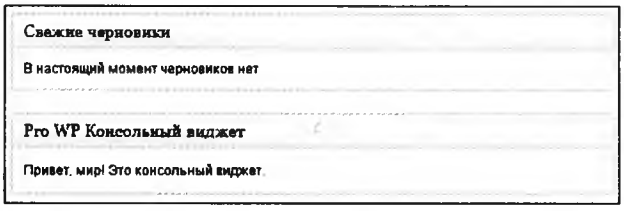


Рис. 8.6. Пример консольного виджета

Создание произвольных таблиц

WordPress содержит разнообразные таблицы для хранения данных плагина. Однако вам может понадобиться и парочка произвольных таблиц для хранения таких данных. Это полезно для более сложных плагинов, связанных, например, с электронной коммерцией, которые хранят историю заказов, данные о товарах и инвентаризации и другие данные, доступ к которым осуществляется с помощью семантики базы данных SQL, а не посредством простого ключа, а значения являются парными с таблицей параметров.

Первый шаг в создании произвольной таблицы базы данных — это создание функции установки. Эта функция будет выполняться, когда плагин активирован для создания новой таблицы.

```
<?php
register_activation_hook( __FILE__, 'prowp_install' );
function prowp_install() {
}
?>
```

Теперь, при наличии функции установки, нужно определить имя произвольной таблицы. Помните, что префикс таблицы может быть определен пользователем произвольно в `wp-config.php`, и, как будет оговорено в главе 10, WordPress Multisite

может вставлять данные дополнительного префикса в имена таблицы, поэтому префиксы нужно встроить в имя произвольной таблицы. Чтобы получить префикс таблицы, используйте значение глобальной `$wpdb->prefix`:

```
global $wpdb;
// задаем имя произвольной таблицы
$table_name = $wpdb->prefix .'prowp_data';
```

Этот код сохраняет таблицу с названием `wp_prowp_data` в переменной `$table_name`, если префикс таблицы WordPress задан как `wp_`.

Теперь пора построить SQL-запрос для создания новой таблицы. Мы создадим новый запрос в переменной `$sql` перед его выполнением. Также необходимо включить файл `upgrade.php` до того, как выполнять запрос:

```
$sql = "CREATE TABLE " .$table_name ." (
    id mediumint(9) NOT NULL AUTO_INCREMENT,
    time bigint(11) DEFAULT '0' NOT NULL,
    name tinytext NOT NULL,
    text text NOT NULL,
    url VARCHAR(55) NOT NULL,
    UNIQUE KEY id (id)
);";
require_once( ABSPATH . 'wp-admin/includes/upgrade.php' );
// выполняем запрос для создания таблицы
dbDelta( $sql );
```

После выполнения этого кода в базе данных создается новая таблица. Функция `dbDelta()` сначала подтверждает, что создаваемой таблицы не существует, поэтому не нужно беспокоиться о предварительной проверке этого факта. Также неплохо сохранить номер версии для структуры таблицы базы данных. Это может помочь в случае обновления плагина и необходимости изменения структуры таблицы. Вы можете проверить, какую версию таблицы пользователи установили для вашего плагина, и определить, нуждается ли она в обновлении:

```
$prowp_db_version = '1.0';
add_option( 'prowp_db_version', $prowp_db_version );
```

Посмотрим на всю функцию в действии:

```
register_activation_hook( __FILE__, 'prowp_install' );
function prowp_install() {
    global $wpdb;

    // задаем имя произвольной таблицы
    $table_name = $wpdb->prefix .'prowp_data';
    // создаем запрос для создания новой таблицы
    $sql = "CREATE TABLE " .$table_name ." (
            id mediumint(9) NOT NULL AUTO_INCREMENT,
            time bigint(11) DEFAULT '0' NOT NULL,
            name tinytext NOT NULL,
            text text NOT NULL,
            url VARCHAR(55) NOT NULL,
            UNIQUE KEY id (id)
        );";
```

```
    require_once( ABSPATH . 'wp-admin/includes/upgrade.php' );
    // выполняем запрос для создания таблицы
    dbDelta( $sql );
    // устанавливаем версию структуры таблицы
    $prowp_db_version = '1.0';

    // сохраняем номер версии структуры таблицы
    add_option( 'prowp_db_version', $prowp_db_version );
}
```

Если вы хотите обновить структуру таблицы для новой версии плагина, достаточно сравнить номера версий:

```
$installed_ver = get_option( 'gmp_db_version' );
if( $installed_ver != $prowp_db_version ) {
    // обновляем таблицу
    // обновляем версию таблицы
    update_option( 'gmp_db_version', $prowp_db_version );
}
```

Перед созданием произвольной таблицы для плагина необходимо подумать, а лучший ли это метод. В целом следует избегать создания произвольных таблиц, если имеются альтернативы. Помните, что вы можете без проблем сохранять параметры в WordPress, используя API параметров. Вы также можете использовать таблицы `wp_*meta` для хранения расширенных данных по записям, страницам, комментариям и пользователям. Пользовательские типы записей также прекрасно подходят для хранения данных.

Чтобы работать с произвольной таблицей после ее создания, вам нужно использовать класс базы данных WordPress, как показано в главе 6.

Деинсталляция плагина

Весьма неплохо включать в плагин функцию деинсталляции. WordPress поддерживает два варианта регистрации деинсталлятора: метод `uninstall.php` и зацепку-деинсталлятор. Оба метода используются, когда деактивированный плагин удаляется из консоли администратора WordPress.

Первый метод, который мы рассмотрим, — это деинсталлятор `uninstall.php`. Это предпочтительный вариант деинсталляции плагина. Первый шаг на пути его использования — создание файла `uninstall.php`. Он должен существовать в корневой директории плагина. В случае его наличия у него будет приоритет перед зацепкой-деинсталлятором.

```
<?php
// если функция uninstall/delete вызвана не из WordPress, выходим
if( !defined( 'ABSPATH' ) && !defined( 'WP_UNINSTALL_PLUGIN' ) )
    exit();
// удаляем параметр из таблицы параметров
delete_option( 'prowp_options_arr' );
// удаляем все другие параметры, произвольные таблицы и файлы
?>
```

Первое, что должен проверить файл `uninstall.php`, — определены ли постоянные `ABSPATH` и `WP_UNINSTALL_PLUGIN`, то есть вызывались ли они из WordPress. Это мера безопасности, гарантирующая, что данный файл не запускался, кроме как во время процесса деинсталляции плагина. Следующий шаг — удаление любых параметров и произвольных таблиц, созданных плагином. В случае идеального сценария деинсталляции плагин не оставит в базе данных ни единого следа. В предыдущем примере для удаления массива параметров используется `delete_option()`. Помните, что после запуска этой функции все данные произвольного плагина будут уничтожены.

Второй метод деинсталляции плагина — использование зацепки-деинсталлятора. Если при удалении плагина `uninstall.php` не существует, но имеется зацепка-деинсталлятор, плагин будет запущен в последний раз для ее отработки. Вот зацепка-деинсталлятор в действии:

```
<?php
register_uninstall_hook( __FILE__, 'prowp_uninstall_hook' );
function prowp_uninstall_hook() {
    delete_option( 'prowp_options_arr' );
    // удаляем все дополнительные параметры и произвольные таблицы
}
?>
```

Сначала вызывается произвольная функция деинсталляции для правильной деинсталляции параметров плагина. Если вы включили в плагин функциональность деинсталляции, такую как удаление произвольных таблиц и параметров, не забудьте предупредить пользователей, что все данные плагина будут удалены при его удалении.

Разница между этим методом и `register_deactivation_hook` в том, что `register_uninstall_hook` выполняется для удаления деактивированного плагина. `register_deactivation_hook` выполняется для деактивации плагина, это означает, что пользователь может когда-нибудь захотеть снова активировать плагин. Вряд ли вы захотите удалять настройки, если пользователь собирается снова прибегнуть к плагину.

Создание плагина для примера

Теперь, когда вы увидели все многообразие параметров, предоставляемых WordPress для использования в плагинах, можно применить эти знания на практике. Создавая этот пример, мы используем все многообразие возможностей, продемонстрированное в этой главе. К концу раздела вы получите полный исходный код плагина.

Плагин, который вы будете делать, представляет собой простенький магазин с товарами для Хэллоуина (Halloween Store). Цель плагина — создать простой способ добавления товаров в WordPress и их отображения в Halloween Store. Нам понадобятся:

- страница настроек с использованием API настроек;
- виджет для отображения новейших товаров с использованием класса `Widget`;
- поле метаданных для добавления метаданных о продукте;
- поддержка сокращенного кода для простоты отображения данных о товаре в записи;
- поддержка интернационализации на базе функций перевода.

Первый шаг в построении плагина — создание файлов плагина. Для данного плагина у вас будет два файла: `halloween-store.php` и `uninstall.php`. Поскольку файлов два, вам придется сохранить их в отдельной папке для плагина, назвав ее `halloween-store`. Теперь нужно определить заголовок и лицензию плагина.

Для начала вы будете работать с `halloween-store.php`. Сначала вам понадобится определить заголовок плагина:

```
<?php
/*
Plugin Name: Halloween Store
Plugin URI: http://webdevstudios.com/support/wordpress-plugins/
Description: Создаем плагин Halloween Store для отображения информации о продуктах
Version: 1.0
Author: Brad Williams
Author URI: http://webdevstudios.com
License: GPLv2
*/
/*  Copyright 2013  Brad Williams  (email : brad@webdevstudios.com)
    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.
    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/
```

Как видите, вы создали надлежащий заголовок для нового плагина. Поскольку вы будете выпускать плагин, под заголовком расположена лицензия GPL для программного обеспечения.

Теперь вы вызываете функцию `register_activation_hook()` для определения настроек плагина по умолчанию. Помните, что эта функция запускается, когда пользователь активирует плагин в WordPress.

```
// Вызываем функцию, когда активируется плагин
register_activation_hook( __FILE__, 'halloween_store_install' );
function halloween_store_install() {
    // устанавливаем параметры по умолчанию
```

```
    $hween_options_arr = array(
        'currency_sign' => '$'
    );
    // сохраняем параметры по умолчанию
    update_option( 'halloween_options', $hween_options_arr );
}
```

Как видите, плагин будет сохранять массив настроек в единственном параметре с названием `halloween_options`. Когда плагин активируется, вы задаете значение по умолчанию для `currency_sign` как `$`.

Теперь вы вызываете зацепку `init` для регистрации пользовательского типа записи для Products. Так вы будете добавлять продукты в Halloween Store и управлять ими.

```
// Зацепка-действие для инициализации плагина
add_action( 'init', 'halloween_store_init' );
// функция инициализация плагина Halloween Store
function halloween_store_init() {
    //register the products custom post type
    $labels = array(
        'name' => __( 'Products', 'halloween-plugin' ),
        'singular_name' => __( 'Product', 'halloween-plugin' ),
        'add_new' => __( 'Add New', 'halloween-plugin' ),
        'add_new_item' => __( 'Add New Product', 'halloween-plugin' ),
        'edit_item' => __( 'Edit Product', 'halloween-plugin' ),
        'new_item' => __( 'New Product', 'halloween-plugin' ),
        'all_items' => __( 'All Products', 'halloween-plugin' ),
        'view_item' => __( 'View Product', 'halloween-plugin' ),
        'search_items' => __( 'Search Products', 'halloween-plugin' ),
        'not_found' =>  __( 'No products found', 'halloween-plugin' ),
        'not_found_in_trash' => __( 'No products found in Trash',
            'halloween-plugin' ),
        'menu_name' => __( 'Products', 'halloween-plugin' )
      );

    $args = array(
        'labels' => $labels,
        'public' => true,
        'publicly_queryable' => true,
        'show_ui' => true,
        'show_in_menu' => true,
        'query_var' => true,
        'rewrite' => true,
        'capability_type' => 'post',
        'has_archive' => true,
        'hierarchical' => false,
        'menu_position' => null,
        'supports' => array( 'title', 'editor', 'thumbnail', 'excerpt' )
    );

    register_post_type( 'halloween-products', $args );
}
```

Обратите внимание, что каждый переводимый элемент помещается в функцию перевода `__()`. Это позволяет пользователям переводить элементы на любой

нужный им язык. Эти функции перевода будут использоваться по всему коду создаваемого плагина.

Теперь вы создаете страницу настроек Halloween Store. Первый шаг — добавить подпункт для страницы настроек в меню Параметры, используя функцию `add_options_page()`:

```
// зацепка-действие, чтобы добавить пункт меню Продукты
add_action( 'admin_menu', 'halloween_store_menu' );
// создаем подпункт меню Halloween Masks
function halloween_store_menu() {
    add_options_page( __( 'Halloween Store Settings Page',
        'halloween-plugin' ), __( 'Halloween Store Settings',
        'halloween-plugin' ), 'manage_options', 'halloween-store-settings',
        'halloween_store_settings_page' );
}
```

Как видите, эта функция создает подпункт меню. Подпункт Halloween Store будет располагаться в консоли в пункте меню Interface. Вы также задаете настройки таким образом, что этот подпункт виден только администратору.

Теперь вам нужно создать страницу актуальных настроек. Как показано в предыдущем коде, страница настроек Halloween Store запускает произвольную функцию `halloween_store_settings_page()`.

```
// создаем страницу настроек плагина
function halloween_store_settings_page() {

    // извлекаем массив настроек плагина
    $hween_options_arr = get_option( 'halloween_options' );

    // устанавливаем значения переменных из массива
    $hs_inventory = ( ! empty( $hween_options_arr['show_inventory'] ) ) ?
        $hween_options_arr['show_inventory'] : '';
    $hs_currency_sign = $hween_options_arr['currency_sign'];
    ?>
    <div class="wrap">
    <h2><?php _e( 'Halloween Store Options', 'halloween-plugin' ) ?></h2>
    <form method="post" action="options.php">
        <?php settings_fields( 'halloween-settings-group' ); ?>
        <table class="form-table">
            <tr valign="top">
            <th scope="row"><?php _e( 'Show Product Inventory',
                'halloween-plugin' ) ?></th>
            <td><input type="checkbox" name="halloween_options[show_inventory]"
                <?php echo checked( $hs_inventory, 'on' ); ?> /></td>
            </tr>
            <tr valign="top">
            <th scope="row"><?php _e( 'Currency Sign', 'halloween-plugin' ) ?></th>
            <td><input type="text" name="halloween_options[currency_sign]"
                value="<?php echo esc_attr( $hs_currency_sign ); ?>"
                size="1" maxlength="1" /></td>
            </tr>
        </table>
        <p class="submit">
```

```
        <input type="submit" class="button-primary"
            value="<?php _e( 'Save Changes', 'halloween-plugin' ); ?>" />
        </p>
    </form>
    </div>
<?php
}
```

У плагина Halloween Store есть два параметра, отвечающих за то, показывать ли наличие товара и какую валюту использовать. Сначала загружается значение массива параметров плагина. Затем переменным задаются значения двух параметров. Используйте трехзначный оператор PHP для определения значения по умолчанию для Inventory. Вы также загружаете в переменную текущее значение валюты. Затем отображается форма страницы настроек с полями обоих параметров. Обратите внимание, что вы используете функцию `settings_fields()` для соединения формы настроек с зарегистрированной настройкой, которая будет определена в коде далее. Это правильный способ сохранения параметров настроек в массиве с использованием API настроек.

После отправки формы WordPress будет использовать API настроек для очистки значений формы и их сохранения в базе данных. Чтобы все заработало, необходимо зарегистрировать функции поля настроек и очистки:

```
// зацепка-действие, чтобы зарегистрировать настройки плагина
add_action( 'admin_init', 'halloween_store_register_settings' );
function halloween_store_register_settings() {
    // регистрируем настройки
    register_setting( 'halloween-settings-group',
        'halloween_options', 'halloween_sanitize_options' );
}
function halloween_sanitize_options( $options ) {
    $options['show_inventory'] = ( ! empty( $options['show_inventory'] ) ) ?
        sanitize_text_field( $options['show_inventory'] ) : '';
    $options['currency_sign'] = ( ! empty( $options['currency_sign'] ) ) ?
        sanitize_text_field( $options['currency_sign'] ) : '';
    return $options;
}
```

Используя функцию `register_setting()`, вы регистрируете группу настроек, `halloween-settings-group`, и имя параметра, `halloween-options`, для использования в форме настроек. Функция `halloween_sanitize_options()` используется для очистки вводимых пользователем данных для каждой настройки перед ее сохранением в WordPress. Это крайне важный в отношении безопасности шаг: убедиться, что отправляемые данные должным образом очищены перед их сохранением в базе данных.

Теперь, когда настройки плагина сохранены, время зарегистрировать метаполе для метаданных товара:

```
// зацепка-действие, чтобы зарегистрировать метаполе товара
add_action( 'add_meta_boxes', 'halloween_store_register_meta_box' );
function halloween_store_register_meta_box() {
```

```
    // создаем произвольное метаполе
    add_meta_box( 'halloween-product-meta',
        __( 'Product Information','halloween-plugin' ),
        'halloween_meta_box', 'halloween-products', 'side', 'default' );
}
```

Используя зацепку-действие `add_meta_boxes`, вызовите произвольную функцию для регистрации метаполя товара. Функция `add_meta_box()` используется для проведения актуальной регистрации. Теперь, когда метаполе зарегистрировано, нужно построить форму метаполя:

```
// создаем метаполе Продукты
function halloween_meta_box( $post ) {
    // извлекаем значения произвольного метаполя
    $hween_sku = get_post_meta( $post->ID, '_halloween_product_sku', true );
    $hween_price = get_post_meta( $post->ID, '_halloween_product_price', true );
    $hween_weight = get_post_meta( $post->ID, '_halloween_product_weight', true );
    $hween_color = get_post_meta( $post->ID, '_halloween_product_color', true );
    $hween_inventory = get_post_meta( $post->ID, '_halloween_product_inventory',
        true );
    // проверяем временное значение из соображений безопасности
    wp_nonce_field( 'meta-box-save', 'halloween-plugin' );

    // отображаем форму метаполя
    echo '<table>';
    echo '<tr>';
    echo '<td>' .__('Sku', 'halloween-plugin').':</td>
        <td><input type="text" name="halloween_product_sku"
        value="'.esc_attr( $hween_sku ).'" size="10"></td>';
    echo '</tr><tr>';
    echo '<td>' .__('Price', 'halloween-plugin').':</td>
        <td><input type="text" name="halloween_product_price"
        value="'.esc_attr( $hween_price ).'" size="5"></td>';
    echo '</tr><tr>';
    echo '<td>' .__('Weight', 'halloween-plugin').':</td>
        <td><input type="text" name="halloween_product_weight"
        value="'.esc_attr( $hween_weight ).'" size="5"></td>';
    echo '</tr><tr>';
    echo '<td>' .__('Color', 'halloween-plugin').':</td>
        <td><input type="text" name="halloween_product_color"
        value="'.esc_attr( $hween_color ).'" size="5"></td>';
    echo '</tr><tr>';
    echo '<td>Inventory:</td>
        <td><select name="halloween_product_inventory"
        id="halloween_product_inventory">
            <option value="In Stock"'
                .selected( $hween_inventory, 'In Stock', false ). '>'
                .__( 'In Stock', 'halloween-plugin' ). '</option>
            <option value="Backordered"'
                .selected( $hween_inventory, 'Backordered', false ). '>'
                .__( 'Backordered', 'halloween-plugin' ). '</option>
            <option value="Out of Stock"'
                .selected( $hween_inventory, 'Out of Stock', false ). '>'
                .__( 'Out of Stock', 'halloween-plugin' ). '</option>
            <option value="Discontinued"'
                .selected( $hween_inventory, 'Discontinued', false ). '>'
```

```
                .__( 'Discontinued', 'halloween-plugin' ). '</option>
        </select></td>';
    echo '</tr>';
    // отображаем легенду раздела с сокращенным кодом
    echo '<tr><td colspan="2"><hr></td></tr>';
    echo '<tr><td colspan="2"><strong>'
        .__( 'Shortcode Legend', 'halloween-plugin' ).'</strong></td></tr>';
    echo '<tr><td>' .__( 'Sku', 'halloween-plugin' ) .':
        </td><td>[hs show=sku]</td></tr>';
    echo '<tr><td>' .__( 'Price', 'halloween-plugin' ).':
        </td><td>[hs show=price]</td></tr>';
    echo '<tr><td>' .__( 'Weight', 'halloween-plugin' ).':
        </td><td>[hs show=weight]</td></tr>';
    echo '<tr><td>' .__( 'Color', 'halloween-plugin' ).':
        </td><td>[hs show=color]</td></tr>';
    echo '<tr><td>' .__( 'Inventory', 'halloween-plugin' ).':
        </td><td>[hs show=inventory]</td></tr>';
    echo '</table>';
}
```

Плагин Halloween Store сохраняет пять различных значений для каждого товара: единицы, цена, вес, цвет и наличие. Как видите, первый шаг — загрузка этих пяти значений произвольного поля. Затем нужно отобразить форму метаполя и заполнить значения валюты, если они существуют. Ниже формы метаполя вы отображаете простое описание коротких кодов, чтобы показать пользователю, какие сокращенные коды можно использовать для отображения метаданных товара. По завершении процесса произвольное метаполе будет выглядеть, как на рис. 8.7.

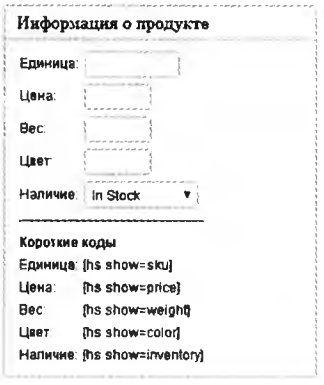


Рис. 8.7. Метаполе записи о продукте

Теперь, когда вы создали произвольное метаполе, нужно сохранить введенные в форму данные:

```
// зацепка-действие для сохранения данных метаполя, когда сохраняется запись
add_action( 'save_post','halloween_store_save_meta_box' );
// сохраняем данные метаполя
function halloween_store_save_meta_box( $post_id ) {
```

```
    // проверяем, относится ли запись к типу Halloween Products
    // и были ли отправлены метаданные
    if ( get_post_type( $post_id ) == 'halloween-products'
        && isset( $_POST['halloween_product_sku'] ) ) {

        // если установлено автосохранение, пропускаем сохранение данных
        if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE )
            return;
        // проверка из соображений безопасности
        check_admin_referer( 'meta-box-save', 'halloween-plugin' );
        // сохраняем данные метаполя в произвольных полях записи
        update_post_meta( $post_id, '_halloween_product_sku',
            sanitize_text_field( $_POST['halloween_product_sku'] ) );
        update_post_meta( $post_id, '_halloween_product_price',
            sanitize_text_field( $_POST['halloween_product_price'] ) );
        update_post_meta( $post_id, '_halloween_product_weight',
            sanitize_text_field( $_POST['halloween_product_weight'] ) );
        update_post_meta( $post_id, '_halloween_product_color',
            sanitize_text_field( $_POST['halloween_product_color'] ) );
        update_post_meta( $post_id, '_halloween_product_inventory',
            sanitize_text_field( $_POST['halloween_product_inventory'] ) );
    }

}
```

Сначала необходимо убедиться, что сохраняемая запись — это запись пользовательского типа `halloween-products`. Также нужно удостовериться, что значение `$_POST['halloween_product_sku']` было задано до начала обработки. Единственное обязательное поле — это единицы измерения товара; если оно пустое, данные товара сохранены не будут. После того как вы убедились, что единицы существуют, нужно удостовериться, что запись не является сохраняемой автоматически. Вам также нужно проверить временное значение для безопасного использования `check_admin_referer()`. После осуществления всех проверок вы сохраняете произвольные поля товара как метаданные для товара, который вы создаете или обновляете.

Затем необходимо настроить сокращенный код для использования плагина. Это позволит вам легко отображать любые метаданные товара в контенте Продукта.

```
// зацепка-действие, чтобы создать сокращенный код для товаров
add_shortcode( 'hs', 'halloween_store_shortcode' );
// создаем сокращенный код
function halloween_store_shortcode( $atts, $content = null ) {
    global $post;

    extract( shortcode_atts( array(
        "show" => ''
    ), $atts ) );
    // извлекаем настройки
    $hween_options_arr = get_option( 'halloween_options' );
    if ( $show == 'sku') {

        $hs_show = get_post_meta( $post->ID, '_halloween_product_sku', true );
```

```
    }elseif ( $show == 'price' ) {

        $hs_show = $hween_options_arr['currency_sign'].
            get_post_meta( $post->ID, '_halloween_product_price', true );

    }elseif ( $show == 'weight' ) {

        $hs_show = get_post_meta( $post->ID,
            '_halloween_product_weight', true );

    }elseif ( $show == 'color' ) {

        $hs_show = get_post_meta( $post->ID,
            '_halloween_product_color', true );

    }elseif ( $show == 'inventory' ) {

        $hs_show = get_post_meta( $post->ID,
            '_halloween_product_inventory', true );

    }
    // возвращаем значение сокращенного кода для отображения
    return $hs_show;
}
```

Первое, что вы делаете, — инициализируете глобальную переменную `$post`. Это даст значение `$post->ID` для записи, в которой вы используете сокращенный код. Затем вы извлекаете определенные вами атрибуты сокращенного кода, в данном случае `show`. Наконец, вы проверяете, какое значение атрибута посылается сокращенному коду для определения того, какое значение показывать. Использование сокращенного кода типа `[hs show=price]` даст отображение цены товара. При отображении ценовых метаданных вам нужно вернуть значение параметра знака валюты, заданное пользователем.

Затем вы создаете виджет товара:

```
// зацепка-действие, чтобы создать виджет
add_action( 'widgets_init', 'halloween_store_register_widgets' );
// регистрируем виджет
function halloween_store_register_widgets() {
    register_widget( 'hs_widget' );
}
//hs_widget class
class hs_widget extends WP_Widget {
```

Сначала нужно зарегистрировать виджет как `hs_widget`, используя функцию `register_widget()`. Затем вы расширяете класс `Widget` как `hs_widget`. Теперь нужно создать четыре функции виджета, необходимые для его построения:

```
    // создаем виджет
    function hs_widget() {

        $widget_ops = array(
            'classname'   => 'hs-widget-class',
```

```
        'description' => __( 'Display Halloween Products',
            'halloween-plugin' ) );
    $this->WP_Widget( 'hs_widget', __( 'Products Widget',
        'halloween-plugin'), $widget_ops );

}
```

Первой вы создаете функцию `hs_widget()`, ее часто называют *конструктором*. В ней вы задаете название виджета, описание и класс имени для произвольного виджета:

```
// создаем форму настроек виджета
function form( $instance ) {

    $defaults = array(
        'title'           => __( 'Products', 'halloween-plugin' ),
        'number_products' => '3' );

    $instance = wp_parse_args( (array) $instance, $defaults );
    $title = $instance['title'];
    $number_products = $instance['number_products'];
    ?>
        <p><?php _e('Title', 'halloween-plugin') ?>:
            <input class="widefat"
                name="<?php echo $this->get_field_name( 'title' ); ?>"
                type="text" value="<?php echo esc_attr( $title ); ?>" /></p>
        <p><?php _e( 'Number of Products', 'halloween-plugin' ) ?>:
            <input name="
                <?php echo $this->get_field_name( 'number_products' ); ?>"
                type="text" value="<?php echo esc_attr( $number_products ); ?>"
                size="2" maxlength="2" />
        </p>
    <?php
}
```

Вторая функция, которую вы определяете, — это `form()`. В виджете вы сохраняете две настройки: название виджета и количество отображаемых товаров. Сначала нужно определить значения по умолчанию на случай, если никакие значения не сохранены. Затем следует загрузить сохраненные значения для двух настроек. Наконец, отображаются оба поля формы настроек со значениями настроек, если таковые существуют.

```
// сохраняем настройки виджета
function update( $new_instance, $old_instance ) {
    $instance = $old_instance;
    $instance['title'] = sanitize_text_field( $new_instance['title'] );
    $instance['number_products'] = absint( $new_instance['number_products'] );
    return $instance;
}
```

После этого вы создаете функцию `update()`. Она сохраняет настройки виджета. Обратите внимание, как используется функция `sanitize_text_field()` для очистки названия виджета. Вы также используете функцию PHP `absint()`, чтобы убедиться, что значение количества товаров является неотрицательным целым.

```
    // отображаем виджет
    function widget( $args, $instance ) {
        global $post;

        extract( $args );
        echo $before_widget;
        $title = apply_filters( 'widget_title', $instance['title'] );
        $number_products = $instance['number_products'];
        if ( ! empty( $title ) ) { echo $before_title . esc_html( $title )
            . $after_title; };
        // произвольный запрос, чтобы извлечь продукты
        $args = array(
            'post_type'         =>    'halloween-products',
            'posts_per_page'    =>    absint( $number_products )
        );

        $dispProducts = new WP_Query();
        $dispProducts->query( $args );

        while ( $dispProducts->have_posts() ) : $dispProducts->the_post();
            // извлекаем массив настроек
            $hween_options_arr = get_option( 'halloween_options' );
            // извлекаем значения произвольных полей
            $hs_price = get_post_meta( $post->ID,
                '_halloween_product_price', true );
            $hs_inventory = get_post_meta( $post->ID,
                '_halloween_product_inventory', true );
            ?>
            <p>
                <a href="<?php the_permalink(); ?>"
                    rel="bookmark"
                    title="<?php the_title_attribute(); ?> Product Information">
                <?php the_title(); ?>
                </a>
            </p>
            <?php
            echo '<p>' .__( 'Price', 'halloween-plugin' ). ': '
                .$hween_options_arr['currency_sign'] .$hs_price .'</p>';
            // проверяем, включена ли опция отображения списка продуктов
            if ( $hween_options_arr['show_inventory'] ) {

                // отображаем метаданные для продукта
                echo '<p>' .__( 'Stock', 'halloween-plugin' ). ': '
                    .$hs_inventory .'</p>';

            }
            echo '<hr>';
        endwhile;

        wp_reset_postdata();
        echo $after_widget;

    }
}
```

Последняя определяемая функция — `widget()`. Эта функция отображает виджет в публичном доступе на сайте. Сначала инициализируйте глобальную переменную

`$post` и извлеките `$args` для виджета. Затем отобразите переменную `$before_widget`. Она может задаваться разработчиками темы и плагина для отображения определенного контента до и после плагина. После этого вы возвращаете значения двух настроек. Если значение `$title` не пустое, используется оно, если же пустое, используется заданное ранее название по умолчанию.

Чтобы отобразить в виджете товары, создайте произвольный цикл с помощью `WP_Query`, как описано в главе 5. Помните, что поскольку это не основной цикл, нужно использовать для его создания `WP_Query`, а не `query_posts()`. Чтобы определить произвольный цикл, передайте два параметра: один для типа записи и один для числа отображаемых товаров. Первое значение (`post_type=halloween-products`) говорит произвольному циклу о необходимости возвращать только информацию по товарам Halloween. Второе значение, `posts_per_page`, определяет, сколько товаров отображать. Это число извлекается из значения параметров виджета, задаваемого пользователем.

Затем загрузите значения параметра и значения произвольных метаданных, которые будут отображаться в виджете. Наконец, вы отображаете в виджете значения товаров. Если параметр «Показать наличие» активирован, то будет выводиться информация о наличии. После успешного создания виджет «Продукты» будет выглядеть, как показано на рис. 8.8.

Рис. 8.8. Виджет «Продукты»

Последний шаг в создании плагина Halloween Store — это файл `uninstall.php`:

```
<?php
// выходим, если uninstall/delete вызвана не из WordPress
if( ! defined( 'ABSPATH' ) && ! defined( 'WP_UNINSTALL_PLUGIN' ) )
    exit ();
// удаляем данные из таблицы параметров
delete_option( 'halloween_options' );
?>
```

Первое, что следует проверить, — это наличие постоянных `ABSPATH` и `WP_UNINSTALL_PLUGIN`. Это значит, что они вызывались из WordPress и добавляют еще один слой безопасности в деинсталлятор. После того как вы убедились в правомочности запроса, удалите значение единичного параметра из базы данных. Здесь также можно

при необходимости определить другие функциональности деинсталляции, такие как удаление каждого значения метаданных товара, сохраненных в базе данных.

Готово! Вы только что успешно построили полноценный плагин, включающий в себя многие из возможностей, описанных в этой части. Это самый базовый плагин, но он дает вам примеры и инструментарий, необходимые для дальнейшей работы. В листинге 8.5 показан полный исходный код плагина. Чтобы получить этот код онлайн, зайдите по ссылке https://github.com/williamsba/HalloweenStore.

Листинг 8.5. Полный исходный код плагина (halloween-store.zip)

```
<?php
/*
Plugin Name: Halloween Store
Plugin URI: http://webdevstudios.com/support/wordpress-plugins/
Description: Создаем плагин Halloween Store для отображения информации о товарах
Version: 1.0
Author: Brad Williams
Author URI: http://webdevstudios.com
License: GPLv2
*/
/*  Copyright 2013  Brad Williams  (email : brad@webdevstudios.com)
    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.
    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/
// Вызываем функцию, когда активируется плагин
register_activation_hook( __FILE__, 'halloween_store_install' );
function halloween_store_install() {
    // устанавливаем параметры по умолчанию
    $hween_options_arr = array(
        'currency_sign' => '$'
    );
    // сохраняем параметры по умолчанию
    update_option( 'halloween_options', $hween_options_arr );
}
// зацепка-действие для инициализации плагина
add_action( 'init', 'halloween_store_init' );
// инициализируем Halloween Store
function halloween_store_init() {
    // регистрируем пользовательский тип записи для товаров
    $labels = array(
        'name' => __( 'Products', 'halloween-plugin' ),
        'singular_name' => __( 'Product', 'halloween-plugin' ),
        'add_new' => __( 'Add New', 'halloween-plugin' ),
        'add_new_item' => __( 'Add New Product', 'halloween-plugin' ),
        'edit_item' => __( 'Edit Product', 'halloween-plugin' ),
```

```
        'new_item' => __( 'New Product', 'halloween-plugin' ),
        'all_items' => __( 'All Products', 'halloween-plugin' ),
        'view_item' => __( 'View Product', 'halloween-plugin' ),
        'search_items' => __( 'Search Products', 'halloween-plugin' ),
        'not_found' =>  __( 'No products found', 'halloween-plugin' ),
        'not_found_in_trash' => __( 'No products found in Trash',
            'halloween-plugin' ),
        'menu_name' => __( 'Products', 'halloween-plugin' )
      );

      $args = array(
        'labels' => $labels,
        'public' => true,
        'publicly_queryable' => true,
        'show_ui' => true,
        'show_in_menu' => true,
        'query_var' => true,
        'rewrite' => true,
        'capability_type' => 'post',
        'has_archive' => true,
        'hierarchical' => false,
        'menu_position' => null,
        'supports' => array( 'title', 'editor', 'thumbnail', 'excerpt' )
      );

      register_post_type( 'halloween-products', $args );
}
// зацепка-действие, чтобы добавить пункт меню товаров
add_action( 'admin_menu', 'halloween_store_menu' );
// создаем подменю Halloween Masks
function halloween_store_menu() {

    add_options_page( __( 'Halloween Store Settings Page', 'halloween-plugin' ),
    __( 'Halloween Store Settings', 'halloween-plugin' ),
    'manage_options', 'halloween-store-settings',
    'halloween_store_settings_page' );

}
// создаем страницу настроек плагина
function halloween_store_settings_page() {

    // извлекаем массив настроек плагина
    $hween_options_arr = get_option( 'halloween_options' );

    // устанавливаем переменные значения массива
    $hs_inventory = ( ! empty( $hween_options_arr['show_inventory'] ) )
        ? $hween_options_arr['show_inventory'] : '';
    $hs_currency_sign = $hween_options_arr['currency_sign'];
    ?>
    <div class="wrap">
    <h2><?php _e( 'Halloween Store Options', 'halloween-plugin' ) ?></h2>
    <form method="post" action="options.php">
        <?php settings_fields( 'halloween-settings-group' ); ?>
        <table class="form-table">
            <tr valign="top">
            <th scope="row"><?php _e( 'Show Product Inventory',
```

```
            'halloween-plugin' ) ?></th>
        <td><input type="checkbox" name="halloween_options[show_inventory]"
        <?php echo checked( $hs_inventory, 'on' ); ?> /></td>
        </tr>
        <tr valign="top">
        <th scope="row"><?php _e( 'Currency Sign', 'halloween-plugin' ) ?></th>
        <td><input type="text" name="halloween_options[currency_sign]"
        value="<?php echo esc_attr( $hs_currency_sign ); ?>"
        size="1" maxlength="1" /></td>
        </tr>
    </table>
    <p class="submit">
    <input type="submit" class="button-primary"
    value="<?php _e( 'Save Changes', 'halloween-plugin' ); ?>" />
    </p>
</form>
</div>
<?php
}
// зацепка-действие, чтобы зарегистрировать настройки плагина
add_action( 'admin_init', 'halloween_store_register_settings' );
function halloween_store_register_settings() {

    // регистрируем массив настроек
    register_setting( 'halloween-settings-group',
        'halloween_options', 'halloween_sanitize_options' );

}
function halloween_sanitize_options( $options ) {

    $options['show_inventory'] = ( ! empty( $options['show_inventory']
        ) ) ? sanitize_text_field( $options['show_inventory'] ) : '';
    $options['currency_sign'] = ( ! empty( $options['currency_sign']
        ) ) ? sanitize_text_field( $options['currency_sign'] ) : '';

    return $options;

}
// зацепка-действие, чтобы зарегистрировать метаполе товаров
add_action( 'add_meta_boxes', 'halloween_store_register_meta_box' );
function halloween_store_register_meta_box() {

    // создаем произвольное метаполе
    add_meta_box( 'halloween-product-meta',
        __( 'Product Information','halloween-plugin' ),
        'halloween_meta_box', 'halloween-products', 'side',
        'default' );

}
// создаем метаполе товаров
function halloween_meta_box( $post ) {
    // извлекаем значения произвольного метаполя
    $hween_sku = get_post_meta( $post->ID, '_halloween_product_sku', true );
    $hween_price = get_post_meta( $post->ID, '_halloween_product_price', true );
    $hween_weight = get_post_meta( $post->ID, '_halloween_product_weight', true );
    $hween_color = get_post_meta( $post->ID, '_halloween_product_color', true );
```

```
$hween_inventory = get_post_meta( $post->ID,
'_halloween_product_inventory', true );
// проверяем из соображений безопасности
wp_nonce_field( 'meta-box-save', 'halloween-plugin' );

// отображаем форму метаполя
echo '<table>';
echo '<tr>';
echo '<td>' .__('Sku', 'halloween-plugin').':</td><td>
    <input type="text" name="halloween_product_sku"
    value="'.esc_attr( $hween_sku ).'" size="10"></td>';
echo '</tr><tr>';
echo '<td>' .__('Price', 'halloween-plugin').':</td><td>
    <input type="text" name="halloween_product_price"
    value="'.esc_attr( $hween_price ).'" size="5"></td>';
echo '</tr><tr>';
echo '<td>' .__('Weight', 'halloween-plugin').':</td><td>
    <input type="text" name="halloween_product_weight"
    value="'.esc_attr( $hween_weight ).'" size="5"></td>';
echo '</tr><tr>';
echo '<td>' .__('Color', 'halloween-plugin').':</td><td>
    <input type="text" name="halloween_product_color"
    value="'.esc_attr( $hween_color ).'" size="5"></td>';
echo '</tr><tr>';
echo '<td>Inventory:</td><td>
    <select name="halloween_product_inventory"
        id="halloween_product_inventory">
        <option value="In Stock"'
        .selected( $hween_inventory, 'In Stock', false ). '>'
            .__( 'In Stock', 'halloween-plugin' )
        . '</option>
        <option value="Backordered"'
        .selected( $hween_inventory, 'Backordered', false ). '>'
            .__( 'Backordered', 'halloween-plugin' )
        . '</option>
        <option value="Out of Stock"'
        .selected( $hween_inventory, 'Out of Stock', false )
        . '>' .__( 'Out of Stock', 'halloween-plugin' ). '</option>
        <option value="Discontinued"'
        .selected( $hween_inventory, 'Discontinued', false )
        . '>' .__( 'Discontinued', 'halloween-plugin' ). '</option>
    </select></td>';
echo '</tr>';
// отображаем легенду раздела с сокращенным кодом
echo '<tr><td colspan="2"><hr></td></tr>';
echo '<tr><td colspan="2"><strong>' .__( 'Shortcode Legend',
    'halloween-plugin' ).'</strong></td></tr>';
echo '<tr><td>' .__( 'Sku', 'halloween-plugin' ) .':</td><td>
     [hs show=sku]</td></tr>';
echo '<tr><td>' .__( 'Price', 'halloween-plugin' ).':</td><td>
     [hs show=price]</td></tr>';
echo '<tr><td>' .__( 'Weight', 'halloween-plugin' ).':</td><td>
     [hs show=weight]</td></tr>';
echo '<tr><td>' .__( 'Color', 'halloween-plugin' ).':</td><td>
     [hs show=color]</td></tr>';
echo '<tr><td>' .__( 'Inventory', 'halloween-plugin' ).':</td><td>
```

```
        [hs show=inventory]</td></tr>';
    echo '</table>';
}
// зацепка-действие для сохранения данных метаполя, когда сохраняется запись
add_action( 'save_post','halloween_store_save_meta_box' );
// сохраняем данные метаполя
function halloween_store_save_meta_box( $post_id ) {
    // проверяем, относится ли запись к типу Halloween Products
    // и были ли метаданные отправлены
    if ( get_post_type( $post_id ) == 'halloween-products'
        && isset( $_POST['halloween_product_sku'] ) ) {

        // если включено автосохранение, пропускаем сохранение данных
        if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE )
            return;
        // проверяем из соображений безопасности
        check_admin_referer( 'meta-box-save', 'halloween-plugin' );
        // сохраняем данные метаполя в произвольных полях записи
        update_post_meta( $post_id, '_halloween_product_sku',
            sanitize_text_field( $_POST['halloween_product_sku'] ) );
        update_post_meta( $post_id, '_halloween_product_price',
            sanitize_text_field( $_POST['halloween_product_price'] ) );
        update_post_meta( $post_id, '_halloween_product_weight',
            sanitize_text_field( $_POST['halloween_product_weight'] ) );
        update_post_meta( $post_id, '_halloween_product_color',
            sanitize_text_field( $_POST['halloween_product_color'] ) );
        update_post_meta( $post_id, '_halloween_product_inventory',
            sanitize_text_field( $_POST['halloween_product_inventory'] ) );
    }

}
// зацепка-действие для создания сокращенного кода товаров
add_shortcode( 'hs', 'halloween_store_shortcode' );
// создаем сокращенный код
function halloween_store_shortcode( $atts, $content = null ) {
    global $post;

    extract( shortcode_atts( array(
        "show" => ''
    ), $atts ) );
    // извлекаем массив настроек
    $hween_options_arr = get_option( 'halloween_options' );
    if ( $show == 'sku') {

        $hs_show = get_post_meta( $post->ID, '_halloween_product_sku', true );

    }elseif ( $show == 'price' ) {

        $hs_show = $hween_options_arr['currency_sign']. get_post_meta( $post->ID,
        '_halloween_product_price', true );

    }elseif ( $show == 'weight' ) {

        $hs_show = get_post_meta( $post->ID, '_halloween_product_weight', true );

    }elseif ( $show == 'color' ) {
```

```
        $hs_show = get_post_meta( $post->ID, '_halloween_product_color', true );

    }elseif ( $show == 'inventory' ) {

        $hs_show = get_post_meta( $post->ID,
            '_halloween_product_inventory', true );

    }
    // возвращаем значение сокращенного кода для отображения
    return $hs_show;
}
// зацепка-действие, чтобы создать виджет
add_action( 'widgets_init', 'halloween_store_register_widgets' );
// регистрируем виджет
function halloween_store_register_widgets() {

    register_widget( 'hs_widget' );

}
//hs_widget class
class hs_widget extends WP_Widget {
    // создаем новый виджет
    function hs_widget() {

        $widget_ops = array(
            'classname'   => 'hs-widget-class',
            'description' =>
             __( 'Display Halloween Products','halloween-plugin' ) );
        $this->WP_Widget( 'hs_widget', __( 'Products Widget','halloween-plugin'),
            $widget_ops );

    }
    // создаем форму настроек виджета
    function form( $instance ) {

        $defaults = array(
            'title'           => __( 'Products', 'halloween-plugin' ),
            'number_products' => '3' );

        $instance = wp_parse_args( (array) $instance, $defaults );
        $title = $instance['title'];
        $number_products = $instance['number_products'];
        ?>
            <p><?php _e('Title', 'halloween-plugin') ?>:
                <input class="widefat" name="<?php echo $this->get_field_name(
                'title' ); ?>"
            type="text" value="<?php echo esc_attr( $title ); ?>" /></p>
            <p><?php _e( 'Number of Products', 'halloween-plugin' ) ?>:
                <input name="<?php echo $this->get_field_name(
                'number_products' ); ?>"
            type="text" value="<?php echo esc_attr( $number_products ); ?>"
            size="2" maxlength="2" />
            </p>
        <?php
    }
```

```
// сохраняем настройки виджета
function update( $new_instance, $old_instance ) {

    $instance = $old_instance;
    $instance['title'] =
        sanitize_text_field( $new_instance['title'] );
    $instance['number_products'] =
        absint( $new_instance['number_products'] );
    return $instance;

}
 // отображаем виджет
function widget( $args, $instance ) {
    global $post;

    extract( $args );
    echo $before_widget;
    $title = apply_filters( 'widget_title', $instance['title'] );
    $number_products = $instance['number_products'];
    if ( ! empty( $title ) ) { echo $before_title . esc_html( $title )
        . $after_title; };
    // произвольный запрос для извлечения товаров
    $args = array(
        'post_type'            =>    'halloween-products',
        'posts_per_page'    =>    absint( $number_products )
    );

    $dispProducts = new WP_Query();
    $dispProducts->query( $args );

    while ( $dispProducts->have_posts() ) : $dispProducts->the_post();
        // извлекаем массив настроек
        $hween_options_arr = get_option( 'halloween_options' );
        // извлекаем значения произвольных полей
        $hs_price = get_post_meta( $post->ID,
            '_halloween_product_price', true );
        $hs_inventory = get_post_meta( $post->ID,
            '_halloween_product_inventory', true );
        ?>
        <p>
            <a href="<?php the_permalink(); ?>" rel="bookmark"
            title="<?php the_title_attribute(); ?> Product Information">
            <?php the_title(); ?>
            </a>
        </p>
        <?php
        echo '<p>' .__( 'Price', 'halloween-plugin' ). ': '
            .$hween_options_arr['currency_sign'] .$hs_price .'</p>';
        // проверяем, включена ли опция отображения списка
        if ( $hween_options_arr['show_inventory'] ) {

            // отображаем метаданные продукта
            echo '<p>' .__( 'Stock', 'halloween-plugin' )
            . ': ' .$hs_inventory .'</p>';

        }
        echo '<hr>';
```

```
        endwhile;

        wp_reset_postdata();
        echo $after_widget;

    }

}
```

Публикация в директории плагинов

Пришло время выпустить плагин в мир! Выпуск плагина на WordPress.org не является обязательным требованием, но это лучший способ опубликовать его и дать другим пользователям возможность скачивания и установки. Помните, что директория плагинов на WordPress.org напрямую подключается к любой копии WordPress, поэтому если плагин существует в директории, то любой, кто использует WordPress, может его легко скачать и установить.

Ограничения

Есть несколько ограничений по принятию плагина в директорию плагинов:

- Плагин должен быть совместим с GPLv2 или более поздней версией.
- Плагин не должен нарушать закон или быть аморальным.
- Для хостинга плагина должен использоваться репозиторий Subversion (SVN).
- Плагин не может содержать ссылки на внешние сайты пользователя (такие, как ссылка типа «powered by») без получения предварительного разрешения пользователя плагина.

Убедитесь, что вы соблюдаете эти указания, или же плагин будет удален из директории плагинов.

Загрузка плагина

Первый шаг — создание учетной записи на WordPress.org, если у вас ее еще нет. Для регистрации новой учетной записи зайдите на регистрационную страницу http://wordpress.org/support/register.php. Эта учетная запись WordPress.org будет использоваться как в директории плагинов, так и на форумах поддержки.

После регистрации учетной записи и осуществления входа нужно добавить плагин для включения в директорию плагинов на WordPress.org. Чтобы добавить плагин, пройдите по ссылке Add Your Plugin (Добавить новый плагин) http://wordpress.org/extend/plugins/add/.

Первое требуемое поле — имя плагина (Plugin Name). Имя плагина должно быть в точности таким, какое вы хотите использовать для плагина. Помните, что имя

плагина будет использоваться как URL в директории. Например, если вы добавляете плагин с именем WP Brad, URL плагина в директории плагинов будет http://wordpress.org/extend/plugins/wp-brad/. Как видите, вводимое здесь имя крайне важно и его нельзя изменить.

Второе обязательное поле — описание плагина (Plugin Description). Оно должно содержать детальное описание плагина. Помните, что описание — единственная информация, позволяющая принять решение, будет или не будет ваш плагин разрешен в директории. Ясно изложите функциональность плагина, его цели и инструкции по установке.

Последнее поле — ссылка на плагин (Plugin URL). Это необязательное поле, но включение ссылки для скачивания плагина крайне рекомендуется. Это позволяет обозревателю плагина скачать его и изучить, если нужно. Еще раз: поле не является обязательным, но его заполнение всячески приветствуется.

После того как вы ввели всю информацию, нажмите на кнопку Send Post (Отправить запись), чтобы послать запрос на плагин. Директория плагинов отвечает: «Within some vaguely defined amount of time, someone will approve your request» («В течение некоторого неопределенного промежутка времени кто-нибудь одобрит ваш запрос»). Это не говорит ни о чем, но большинство плагинов одобряются в течение примерно одного дня. То, что ваш плагин одобрен, не означает, что дело сделано. Следующий шаг — загрузка плагина в созданный для этого репозиторий Subversion.

Создание файла readme.txt

Для подачи плагина в директорию плагинов обязательно наличие одного файла: `readme.txt`. Он используется для внесения всей информации по плагину на странице подробностей о плагине в директории. WordPress разработал стандарт файла readme, который точно детализирует, как именно следует определять `readme.txt`. Вот пример файла `readme.txt`:

```
=== Plugin Name ===
Contributors: williamsba1, messenlehner, ericlewis, jtsternberg
Donate link:
http://example.com/donate
Tags: admin, post, images, page, widget
Requires at least: 3.0
Tested up to: 3.5
Stable tag: 1.1.0.0
License: GPLv2
Short description of the plugin with 150 chars max.  No markup here.
== Description ==
This is the long description.  No limit, and you can use Markdown
Additional plugin features
*   Feature 1
*   Feature 2
*   Feature 3
For support visit the [Support Forum](http://example.com/forum/ " Support Forum")
== Installation ==
```

```
1. Upload 'plugin-directory' to the '/wp-content/plugins/' directory
2. Activate the plugin through the 'Plugins' SubPanel in WordPress
3. Place '<?php gmp_custom_function(); ?>' in your theme templates
== Frequently Asked Questions ==
= A question that someone might have =
An answer to that question.
= Does this plugin work with WordPress Multisite? =
Absolutely!  This plugin has been tested and
verified to work on the most current version of WordPress Multisite
== Screenshots ==
1. Screenshot of plugin settings page
2. Screenshot of plugin in action
== Changelog ==
= 1.1 =
* New feature details
* Bug fix details
= 1.0 =
* First official release
== Upgrade Notice ==
= 1.1 =
* Security bug fixed
```

Для получения примера файла `readme.txt` онлайн загляните на http://wordpress.org/extend/plugins/about/readme.txt.

WordPress.org также снабжен валидатором `readme.txt`, чтобы вы могли проверить правильность форматирования файла `readme.txt` перед его отправкой в директорию Subversion. Валидатор доступен по ссылке http://wordpress.org/extend/plugins/about/validator/. Давайте рассмотрим отдельные разделы файла `readme.txt`:

```
=== Plugin Name ===
Contributors: williamsba1, messenlehner, ericlewis, jtsternberg
Donate link: http://example.com/donate
Tags: admin, post, images, page, widget
Requires at least: 3.0
Tested up to: 3.5
Stable tag: 1.1.0.0
License: GPLv2
Short description of the plugin with 150 chars max.  No markup here.
```

Раздел Plugin Name (Имя плагина) — одна из самых важных частей `readme.txt`. В первой строке перечисляются те, кто внес свой вклад в работу над плагином. Это разделяемый запятыми список имен пользователей WordPress.org, которые участвовали в работе над плагином. Ссылка на пожертвования должна представлять собой URL либо на осуществление пожертвования, либо на страницу, на которой объясняется, как именно пользователь может сделать пожертвование в пользу автора плагина. Это отличное место для размещения ссылки на пожертвования через PayPal. Теги представляют собой разделяемый запятыми список тегов, описывающих плагин.

Поле Requires at least (Требует минимум) — это минимальная версия WordPress, необходимая для работы плагина. Если ваш плагин не работает в WordPress ниже версии 3.0, то значением этого поля будет 3.0. Точно так же Tested up to (Проверено

вплоть до) — это последняя версия, для которой был протестирован плагин. Обычно это последняя стабильная версия WordPress. Метка Stable (Стабильно) — важное поле, содержащее текущую версию плагина. Это значение всегда должно соответствовать версии, заданной в заголовке плагина. Последний представляет собой краткое описание плагина, которое должно быть не длиннее 150 символов и не может содержать какую-либо разметку.

```
== Description ==
This is the long description.  No limit, and you can use Markdown
Additional plugin features
*   Feature 1
*   Feature 2
*   Feature 3
For support visit the [Support Forum](http://example.com/forum/ " Support Forum")
```

Раздел Description (Описание) содержит детальное описание плагина. Это информация по умолчанию, отображаемая на странице информации о плагине в директории плагинов. Ограничений по длине описания нет. Можно также использовать неупорядоченные списки, показанные в предшествующем примере, и упорядоченные списки. Вставка ссылок также допустима.

```
== Installation ==
1. Upload 'plugin-directory' to the '/wp-content/plugins/' directory
2. Activate the plugin through the 'Plugins' SubPanel in WordPress
3. Place '<?php prowp_custom_function(); ?>' in your theme templates
```

Раздел Installation (Установка) описывает шаги, необходимые для установки плагина. Если у плагина особые требования по установке, убедитесь, что здесь они перечислены подробно. Также неплохо включить сюда имя функций и сокращенные коды, которые можно использовать в плагине.

```
== Frequently Asked Questions ==
= A question that someone might have =
An answer to that question.
= Does this plugin work with WordPress Multisite? =
Absolutely!  This plugin has been tested and
verified to work on the most current version of WordPress Multisite
```

Раздел FAQ — прекрасное место для списка часто задаваемых вопросов. Он помогает ответить на наиболее распространенные вопросы и избежать запросов по поддержке. Вы можете перечислить множество вопросов с ответами:

```
== Screenshots ==
1. Screenshot of plugin settings page
2. Screenshot of plugin in action
```

Раздел Screenshots (Скриншоты) используется для добавления скриншотов плагина на страницу с подробным описанием плагина. Это двухшаговый процесс. Первый шаг — включение описания каждого скриншота в упорядоченный список. Следующий шаг — помещение файлов изображений в корневую директорию (которая более подробно обсуждается далее). Названия файлов изображений

должны соответствовать номерам в списке. Например, скриншот страницы настроек нужно назвать `screenshot-1.png`. Скриншот плагина в действии нужно назвать `screenshot-2.png`. Разрешенные типы файлов: `png`, `jpg`, `jpeg` и `gif`.

```
== Changelog ==
= 1.1 =
* New feature details
* Bug fix details
= 1.0 =
* First official release
```

Следующий раздел Changelog (Журнал изменений). Важен для указания изменений, внесенных при выпуске новой версии плагина. Весьма полезный раздел для любого, кто хочет обновиться до последней версии. Всегда приятно знать точно, что было добавлено и исправлено, чтобы понять, необходимо ли обновить плагин. При выпуске каждой новой версии через директорию плагинов следует добавлять новый пункт, сколь бы малыми ни были обновления.

```
== Upgrade Notice ==
= 1.1 =
* Security bug fixed
```

Последний раздел — Upgrade Notice (Замечания по обновлению). Этот раздел позволяет добавить особые замечания по обновлению для пользователя WordPress. Сообщения показываются во вкладке **Консоль ▸ Обновления** при выпуске новой версии плагина.

Файл `readme.txt` может содержать и произвольные разделы в том же формате. Это полезно для более сложных плагинов, по которым необходимо предоставлять дополнительную информацию. Произвольные разделы появятся после обязательных, описанных ранее.

Установка SVN

Директория плагинов использует Subversion (SVN) для работы с плагинами. Чтобы опубликовать свой плагин в директории, нужно установить и сконфигурировать клиент SVN. В данном примере мы будем использовать TortoiseSVN для Windows. TortoiseSVN — бесплатный интерфейс-клиент GUI для SVN. Список клиентов SVN для других платформ можно найти по ссылке: http://subversion.apache.org/.

Сначала необходимо скачать соответствующий установщик по ссылке http://tortoisesvn.net/downloads.html. После установки TortoiseSVN необходимо перезагрузить компьютер. Следующий шаг — создание новой директории на компьютере для хранения файлов плагина. Рекомендуется создать поле для хранения всех плагинов типа `c:\projects\wordpress-plugins`. Это существенно облегчит работу, если вы создаете и выпускаете множество плагинов на WordPress.org.

Затем перейдите в директорию `wordpress-plugins` и создайте новую директорию для плагина. Щелкните правой кнопкой мыши на новой папке, чтобы отобразить

контекстное меню. Вы увидите список новых параметров TortoiseSVN: SVN Checkout и ToirtoiseSVN. Выберите SVN Checkout, появится диалоговое окно, как показано на рис. 8.9.

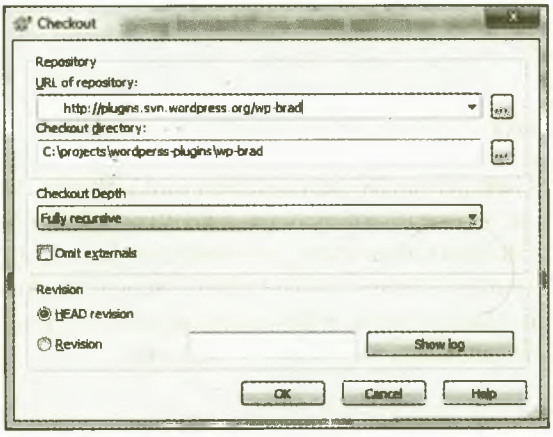


Рис. 8.9. Диалоговое окно SVN Checkout

URL репозитория был прислан вам в электронном письме, которое вы получили после одобрения плагина. Этот URL должен быть таким же, как URL плагина, то есть в данном примере URL будет http://plugins.svn.wordpress.org/wp-brad. Директория Checkout — это локальное поле, в котором хранится плагин. В данном случае вы будете использовать новую созданную вами папку `c:\projects\wordpress-plugins\wp-brad`. Убедитесь, что параметр Checkout Depth задан как Fully Recursive. Также убедитесь, что Revision определена как HEAD Revision. Наконец, нажмите OK. TortoiseSVN подключится к репозиторию SVN для вашего плагина и, если все пойдет нормально, создаст в папке три новые директории: `branches`, `tags` и `trunk`. Каждая из них служит своей цели для SVN:

1. **Branches.** Каждый раз при выпуске новой основной версии появляется branch. Это позволяет исправлять ошибки, не выпуская новую функциональность через `trunk`.
2. **Tags.** Каждый раз при выпуске новой версии для нее создается новая метка.
3. **Trunk.** Основная область разработки. Следующий основной релиз кода живет здесь.

Теперь, подключившись к репозиторию плагина SVN, нужно переместить файлы плагина в директорию `trunk`. Не забудьте поместить в эту директорию для плагина также файл `readme.txt`, скриншоты, включения и пр. Помните, что вы готовите файлы плагина для их публикации в директории плагина. Публикация файлов в WordPress.org освещена в следующем разделе.

После того как вы убедились, что все файлы плагина находятся в `trunk`, вы готовы к публикации плагина в директории плагинов!

Публикация в директории плагинов

Публикация плагина в директории плагинов — двухшаговый процесс. Сначала вам требуется, чтобы SVN зафиксировала папку `trunk` в репозитории SVN. Затем нужно задать метку для версии плагина. По завершении обоих шагов подождите примерно 15 минут, и новый плагин появится в директории плагинов.

Чтобы зафиксировать ветвь плагина, просто щелкните правой кнопкой мыши на папке `trunk` и выберите SVN Commit. Вы увидите диалоговое окно для ввода сообщения об изменениях и выбора файлов для фиксации в `trunk`. Введите краткое сообщение, например «Добавление WP-Brad 1.1», и выберите все файлы, которые вы хотите зафиксировать. TortoiseSVN будет автоматически выбирать все измененные файлы, поэтому, возможно, вам не нужно менять эту настройку. Затем нажмите OK и введите имя пользователя и пароль, которые вы указали на WordPress.org.

Затем нужно задать метку версии плагина. Чтобы сделать это, щелкните правой кнопкой мыши по директории `trunk` и выберите TortoiseSVN ▸ Branch/tag из контекстного меню. В появившемся диалоговом окне укажите путь к снабжаемой меткой директории. Для данного примера URL будет http://plugins.svn.wordpress.org/wp-brad/tags/1.1.0.0/. Метка версии должна соответствовать метке в файле `readme.txt` — в данном случае версия 1.1.0.0. Также напечатайте сообщение об изменениях, например «tagging version 1.1.0.0», и убедитесь что для параметра Create Copy выбрано «HEAD revision in the repository». Нажмите OK, и плагин создаст новую директорию в поле `tags` для версии 1.1.0.0 с соответствующими файлами плагина.

Готово! Если все прошло успешно, через 15 минут плагин появится в директории плагинов. После того как плагин опубликован, следует проверить, что вся информация верна. Один из способов проверить, что плагин успешно опубликован, — это пройти по URL Subversion, в данном случае http://plugins.svn.wordpress.org/wp-brad/. Здесь вы можете убедиться, что директории `trunk` и `tag` загрузились успешно. Через 15 минут вы также можете проверить плагин, зайдя на официальную страницу директории плагинов http://www.wordpress.org/extend/plugins/wp-brad. Если в файл `readme.txt` нужно внести какие-либо изменения, просто отредактируйте его локально в папке `trunk`, щелкните правой кнопкой мыши по файлу и по SVN Commit.

Выпуск новой версии

Прекрасной возможностью плагинов WordPress является простое обновление плагинов в директории плагинов. При выпуске новой версии на всех сайтах WordPress, с которых загружен этот плагин (неважно, активирован он или нет), отображается сообщение о том, что плагин обновился. Пользователь может использовать автоматическое обновление для простого обновления плагина до последней версии. Это особенно важно, если речь идет об исправлениях, касающихся безопасности и помогающих поддерживать надежность работы WordPress.

Чтобы выпустить новую версию плагина, убедитесь, что вы скопировали файлы обновленного плагина в созданную ранее директорию `/trunk`. Эта папка должна

содержать все файлы для обновленной версии плагина. После того как вы убедились, что все обновленные файлы плагина существуют, просто щелкните правой кнопкой мыши на директории `trunk` и выберите SVN Commit. Не забудьте напечатать короткое сообщение об изменениях типа «Выпущена версия 1.2». TortoiseSVN должна заранее выбрать все измененные файлы, но если этого не произошло, выберите все файлы, которые вы хотите опубликовать, и нажмите OK.

Последний шаг — снабжение новой версии меткой. Чтобы сделать это, щелкните правой кнопкой мыши по директории `trunk` и выберите TortoiseSVN ▸ Branch/tag. Для данного примера URL будет http://plugins.svn.wordpress.org/wp-brad/tags/1.2.0.0/. Не забудьте ввести краткое сообщение о внесенных изменениях типа «Снабжение меткой версии 1.2» и нажмите OK. Готово! Новая версия плагина будет опубликована в директории плагинов в течение 15 минут. После этого плагин появится в верхней части списка недавно обновленных плагинов в WordPress.org.

Директория плагинов WordPress — отличный источник вдохновения и сверки при построении произвольных плагинов. Не бойтесь заглянуть в исходный код другого плагина для проверки. Найдите плагин с функциями, схожими с теми, что нужны вам, и посмотрите, как автор структурировал код, или использовал зацепки, или реализовывал свои идеи посредством обработки ядра WordPress.

Резюме

В этой главе изучалась компоновка плагина WordPress, включая необходимый заголовок плагина, содержащий лицензии на программное обеспечение, совместимые с плагином, а также функции его активации и деактивации. Также были охвачены такие важные вещи, как валидация и очистка данных для обеспечения безопасности плагина. Говорилось и об использовании зацепок, параметрах настройки плагина и разных способах интеграции плагинов в WordPress.

Плагины — это только половина возможностей расширения WordPress, позволяющая добавлять на сайт произвольные функции и зависящую от событий обработку. Если вы хотите изменить вид и впечатление, производимое сайтом, или способ, которым WordPress отображает записи, или создать место для сделанных вами виджетов, то нужно расширить WordPress за счет разработки новой темы, о чем и пойдет речь в главе 9.

9 Разработка тем

В этой главе:

- Понимание различных файлов и шаблонов, составляющих тему
- Изменение существующей темы для собственных нужд
- Причины для использования темы проекта или дочерней темы
- Создание новой темы на базе темы Twenty Eleven

Контент — это король, не так ли? Совершенно верно. Только контент привлечет пользователей на ваш сайт и заставит их вернуться. Но даже если ваш контент является лучшим по заданной теме во всем Интернете, вам необходимо представить его читателю, браузеру и поисковым системам так, чтобы им можно было пользоваться.

Здесь в дело вступают темы. Они контролируют внешний вид сайта, включая удобство пользования и способ подачи контента. Они также отвечают за логику, определяющую, какой тип страницы и, следовательно, тип цикла будет использоваться.

В данной главе рассматривается, как установить тему на веб-сайт, анализируются различные аспекты темы и их применение к подаче контента. Вы также изучите различные стратегии создания темы: будет ли эта тема специально разработана для проекта или представляет собой подгонку готовой темы? К концу этой главы вы разберетесь, как функционируют темы, и у вас будет достаточно знаний, позволяющих построить свой собственный проект или же разработать дочернюю тему.

Зачем использовать тему?

Тема, по сути, — лицо веб-сайта. Именно она создает у посетителя первое впечатление. Никто из нас не судит о книге по обложке, но если ценное содержимое сайта

невозможно читать, тяжело найти или оно просто недоступно по какой-то причине, а сайт загружается медленно или откровенно ужасен, вы, вероятно, потеряете посетителя. А терять посетителей не стоит никогда.

Тема делает для веб-сайта многое. Обычно люди думают, что тема — внешний вид сайта. Это впечатление, производимое сайтом за счет определенного стиля. Это ключевой фактор, обеспечивающий сайту индивидуальность и выделяющий его из общей массы. Да, все это и есть тема. Картинка правда лучше тысячи слов.

Но это лишь графический аспект сайта. Тема же много больше, чем средство маркетинга и продвижения бренда. Она всецело обусловливает пользовательский опыт — и не только. Она контролирует, какой контент генерируется, включая условия ошибок. Тема конвертирует контент и способ его подачи в «сырой» HTML, который отправляется браузеру посредством различных шаблонов.

В целом эта глава об использовании темы для структурирования и контроля доставки контента и создания индивидуального образа веб-сайта. У темы есть и другие функции, включая воздействие на пользовательский опыт и оптимизацию для поисковых систем, о чем также говорится в этой главе.

Установка темы

Начиная с 2010 года WordPress ежегодно поставляет новую предустановленную тему. Тема Twenty Ten, выпущенная, как можно догадаться, в 2010-м, была первой, сменившей достопочтенную тему Kubrick, использовавшуюся с 2005-го. В 2011-м Automattic выпустил Twenty Eleven, а в 2012-м в рамках WordPress 3.5 появилась Twenty Twelve. Эти предустановленные темы хороши для использования, но они являются темами по умолчанию, то есть их можно увидеть на многих сайтах в Сети. Уникальность и индивидуальность могут и не входить в ваши задачи, но в данной главе мы исходим из того, что это существенный момент и что вы не хотите использовать готовую тему.

Как заставить WordPress использовать новую тему? Сначала ее нужно найти или сделать. Число различных ресурсов разного качества с темами WordPress бесконечно. Лучше попробовать некоторые темы и посмотреть, как они работают с вашим контентом, соответствуют ли они вашей индивидуальности и отражают ли задачи сайта.

Есть два простых способа активировать новую тему на веб-сайте: традиционная установка по FTP и появившийся в WordPress 2.8 браузер и установщик тем. Браузер тем ограничен и позволяет устанавливать только темы, одобренные директорией тем WordPress на WordPress.org. Это необязательно плохо, поскольку в директории тем много серьезных приятных на вид тем, лицензированных по GPL и бесплатных (два из требований для включения в список директории). Однако директория — это узкий рынок. Загляните на другие сайты, предлагающие интересные темы WordPress, увы — различные по качеству, как бесплатные, так и за деньги. Чтобы установить такие темы не из директории, нужно использовать FTP.

Установка по FTP

Протокол передачи файлов, или FTP, — старый режим передачи файлов с локального компьютера на сервер. Если ваш хост поддерживает его, вы можете использовать безопасную форму передачи, такую как SFTP или SCP, для перемещения файлов, но здесь концепции схожи.

Скачайте пакет темы, которую вы хотите опробовать, на локальный компьютер и извлеките ее в локальную директорию. Если у вас есть доступ к серверу через терминал, вы можете распаковать архив прямо на сервере, чтобы сократить время передачи. Используя FTP-клиент, подключитесь к веб-серверу. Передайте по FTP или скопируйте файлы в директорию тем на сайте: `/example.com/wp-content/themes/`.

Когда файлы темы окажутся на сервере, войдите в консоль вашего сайта WordPress. Выберите вкладку **Внешний вид**, тему для предварительного просмотра, а затем активируйте ее. Теперь сайт использует новую тему.

Установщик темы

В WordPress 2.8 появился новый установщик тем. Сейчас он предлагается как новая вкладка во вкладке **Темы** в консоли. Вкладка **Управление темами** определяет, какие темы в данный момент доступны для данной копии WordPress.

Вторая вкладка — **Установка тем** — позволяет администратору сайта (или кому-либо еще с соответствующими правами) искать и фильтровать онлайн-директорию тем WordPress на предмет существующих опубликованных тем. Все темы в директории соответствуют определенным условиям включения в список. Среди наиболее важных — обязательное лицензирование в соответствии с GPL или аналогами.

Новый установщик тем вполне симпатичный. Вы просто работаете с фильтрами и условиями поиска по директории. Просматривайте иконки скриншотов, пока не найдете то, что вам нравится. Затем вы можете либо нажать на **Установить**, чтобы сделать тему доступной в текущей копии WordPress, либо просмотреть тему в большем формате с разнообразными элементами HTML, отображаемыми, чтобы оценить стиль CSS. Наконец, вы можете прочитать подробную информацию о теме и увидеть пользовательский рейтинг, пройдя по ссылке **Детали**.

В системе разработки, запущенной на Microsoft Windows 7 и WAMP, это просто работает, что несколько обескураживает. Это вопрос разрешений в Webroot. Хотя все это вызывает некоторые вопросы по поводу того, что еще может быть с такой же легкостью установлено на сайте при данном раскладе, сейчас речь идет о машине разработчика, и удобство иметь возможность проверить новые темы перевешивает сомнения.

Использование установщика тем на рабочем сервере для сайта на WordPress на Ubuntu Linux может дать другие результаты. После выбора в меру ужасной темы для проверки установщик тем просит права на FTP, чтобы поместить файлы на сервер. В данном случае — потому что разрешения по безопасности файла на рабочем сервере установлены так, как нужно для работы, и не позволяют делать подобные вещи. Еще раз: есть некоторые сомнения относительно безопасности передачи

прав FTP, требуемых для обработки. Это схоже с тем, как работает обновление ядра и плагинов WordPress. См. главу 13 о безопасности WordPress для получения информации по разрешениям для директории.

Коротко говоря, установщик тем действительно удачен и удобен для разработки в отношении тестирования новых тем, но из-за возможных осложнений, связанных с безопасностью, тщательно обдумайте его использование в рабочей среде. Баланс удобства и безопасности нередко связан со сложным выбором.

Что такое тема?

Что на самом деле составляет тему? Вы представляете, что делают темы, но каким образом и что именно участвует в процессе? Как уже было замечено ранее, тема делает несколько вещей, включая структурирование контента и придание индивидуальности веб-сайту. Это делается посредством комбинации файлов и типов файлов. Вы увидите в теме комбинацию файлов PHP, CSS и JavaScript. В хорошей теме WordPress стиль, являющийся CSS, располагается отдельно от структуры и логики, поддерживаемых файлами PHP. Хотя причины для нарушения правил есть всегда, стремление держать эти две области разделенными улучшит поддерживаемость и эффективность темы. Каждая тема содержит варианты этих файлов, и файлы каждой темы отличаются.

Файлы шаблона

Файлы шаблона — «мясо» вашей темы. Это файлы кода PHP, которые контролируют, какой контент показывать посетителю. WordPress обычно решает, какой файл шаблона использовать, исходя из запрошенного содержимого. Определенные файлы используются для различных заданий. При первом взгляде количество файлов шаблона в теме приводит в растерянность. Хотя все темы разные, в некоторых может быть всего пара файлов, тогда как другая оказывается крайне сложной. После того как вы изучите различные файлы, включенные в тему, мы рассмотрим иерархию шаблонов, являющуюся тем механизмом, который WordPress использует для определения выбираемых шаблонов.

Эта иерархия шаблонов, которая рассматривается в этой главе далее, и многочисленные доступные типы шаблонов могут ошеломлять, когда вы только начинаете проектировать тему, но вы научитесь принимать всю мощь этой установки. Ее гибкость позволяет осуществлять колоссальный контроль над сайтом и тем, что появляется в браузере, в чем и заключается красота и одна из сильнейших сторон WordPress.

CSS

Темы WordPress действительно стараются отделить контент от стиля. Разработчик темы может игнорировать эти указания и создать плохо разделенную тему, но хороший разработчик темы сделает все правильно.

У темы должен быть как минимум один каскадный лист стиля. Первичный лист стиля для темы должен называться `style.css`. Кроме того, первые несколько строк этого листа должны соответствовать определенным указаниям. Эти требования рассматриваются далее в разделе «Основной файл: Style.css» этой главы. WordPress использует данную информацию, чтобы определить, какие темы доступны для сайта WordPress, и заставить их появляться в консоли во вкладке **Внешний вид**. Если заголовок стиля не задан в соответствии со стандартными требованиями WordPress, он не появится в консоли.

Файл стилей является в точности тем, чем кажется. В него помещаются все стили CSS. Как его структурировать и что с ним делать — дело разработчика темы. Разработка CSS одновременно и искусство, и наука, и целая тема, заслуживающая отдельного обсуждения. В этой книге не рассматриваются все лабиринты CSS, но у Wrox есть некоторое количество превосходных изданий по CSS, которые могут помочь вам в данной теме.

Изображения и ресурсы

Тема, вероятно, включает в себя некоторые файлы изображений и другие креативные ресурсы или файлы JavaScript, такие как плагины jQuery. Эти ресурсы используются в теме, чтобы придать веб-сайту особый внешний вид — тот, который хотите *вы*. Как эти файлы структурированы в вашей теме, зависит от вас. Обычно они помещаются в подпапку основной директории темы, такую как `img/`, `images/`, `assets/` или `js/` в зависимости от типа файла. Одна из приятных вещей, касающихся разделения и упаковки тем, заключается в том, что ссылки на изображения могут быть относительными из файла CSS.

Кроме того, на эти креативные ресурсы можно ссылаться из файлов шаблона, используя встроенные функции WordPress, такие как `bloginfo('stylesheet_directory')`. При верной реализации это позволяет делать тему весьма мобильной.

Плагины

Как было рассказано в главе 8, плагины расширяют функциональность сайта. Некоторые темы требуют специфических плагинов, поскольку функциональность — часть индивидуальности темы или же необходимо достичь какой-то определенной цели. Такие плагины могут входить в пакет темы или требовать дополнительного скачивания. Все плагины находятся в папке плагинов. Несмотря на то что плагины не являются непосредственно частью темы, они могут быть необходимы, чтобы тема вела себя так, как задумал ее создатель.

Создание собственной темы

Теперь вы знаете, как установить и активировать тему на сайте, а также что является различными аспектами темы. Время сделать еще один шаг и создать собственную

тему. Вы можете начать с нуля, но почему бы не взгромоздиться на плечи гигантов, начав с готовой с темы, которая выглядит примерно так, как вам нужно? Или же, если вы не можете найти ничего подходящего, начать с каркаса, где самая тяжелая работа уже выполнена? Нет смысла изобретать велосипед, особенно когда вы можете использовать всю силу программного обеспечения с открытым кодом и начать с кода, который уже работает.

Темы проектов или дочерние темы

Перед погружением предлагаем поговорить о стратегиях разработки. По сути, есть два класса тем, используемых в повседневной разработке. Что именно создавать, зависит от объема индивидуализации и специфики выпускаемого проекта. Цель данного раздела не показать, что одно лучше другого. Как разработчик вы должны определить, что соответствует вашим потребностям и задачам.

Давайте сначала обсудим темы проектов. Это единичные (одноцелевые) темы, нередко представляющие собой модификацию существующей темы для соответствия специфическим потребностям и целям одного конкретного проекта. Это могут быть новые темы проектов, построенные с нуля, или же ответвления существующей темы, измененной для целей проекта. Кроме того, есть стартовые темы, представляющие собой основу для тем проектов. Иногда они называются *болванками* (*bare-bones*) или *голыми* (*naked*) темами. У них умышленно мало стилистических и функциональных особенностей, поскольку это буквально чистая страница, с которой можно начинать. Причина, по которой для проекта выбирается такая тема, — в полной свободе в редактировании файлов PHP и шаблона. Однако, делая это, вы теряете возможность обновлять стартовую тему без потери сделанной индивидуализации.

Второй вариант — то, что называется дочерней темой. Дочерняя тема наследует родительскую. Это значит, что вы получаете всю функциональность и стиль родительской темы, а затем перезаписываете некоторые аспекты посредством дочерней темы. К дочерней теме есть два подхода. Если вы нашли тему, которая почти полностью соответствует нуждам вашего проекта, и необходимо внести только косметические изменения или сделать минимальные настройки функциональности, это и есть верный путь.

Другой подход — использовать каркас темы. Каркасы во многом похожи на стартовые темы для тем проектов, за исключением того, что они спроектированы, чтобы быть родительскими темами для дочерних. Они делают базовую работу. Вы создаете дочернюю тему, в которую можно вносить изменения, сохраняя при этом способность обновлять каркас при выходе новых редакций.

Возможно, вы уже догадались, что технически есть и третий подход. Вы можете сначала разработать собственный каркас или родительскую тему, а потом делать дочерние темы для тех или иных проектов.

Стратегия, которую вы выбираете, зависит от целей и нужд вашего проекта. Повторимся: каждый проект индивидуален, как и каждый разработчик. Только вам решать, какой метод подойдет для вас и вашего проекта, и уравновешивать удобство

поддержки с функциональными требованиями. В целом, если вы меняете тему, верным подходом является дочерняя тема. Однако в реальном мире (включая — но не ограничиваясь — сроки и бюджет) темы проектов могут обеспечить необходимую гибкость.

В этой главе мы подробно изучим тему проекта. Имея в своем арсенале основы темы проекта, вы сможете применить эти принципы и к дочерней теме. Мы вернемся к дочерним темам и каркасам в конце главы.

Что искать в стартовой теме

Иногда проще всего найти тему, максимально близкую к тому, что вы задумали, взять ее как стартовую и изменить. Как минимум вы можете добавить свой логотип. Конечно, нужно внимательно отнестись к лицензии темы. Что удобно: все темы в директории тем `wordpress.org` являются GPL, их можно менять как угодно.

Среди моментов, на которые нужно обратить внимание при начале работы с готовой темой:

- лицензирование оригинальной темы;
- качество кода;
- сколько изменений потребуется;
- источники художественных работ для креативных ресурсов.

Вам захочется убедиться, что вы можете вносить изменения в исходную тему, а также оценить качество кода, поскольку именно вы будете вносить в него изменения. Реализует ли тема те презентационные задачи, которые вы ставите для своего сайта, подходит ли она как шаблон, передает ли данные так, как вам это нужно? Нет смысла начинать с темы, которую придется перерабатывать полностью. В таком случае лучше начать с каркаса и создать дочернюю тему. Достаточно ли в теме проработаны стили CSS? Принималась ли во внимание оптимизация для поисковых систем (SEO) при разработке темы? Сколько изменений вам придется внести, чтобы достичь соответствия своим требованиям, и насколько вас устроит конечный результат? Наконец, есть ли в теме оригиналы художественных ресурсов, такие как исходные файлы Photoshop, для внесения изменений? Если нет, нужны ли они или же вы сможете воссоздать все, что вам необходимо?

При разработке новой темы проекта или изменении дочерней темы для себя или для клиента есть о чем подумать. Удобство изменения предустановленной темы — большое искушение, когда нужно быстро создать и запустить сайт. В самом деле, многие сайты делаются именно таким способом, когда клиент выбирает шаблон, в который нужно внести несколько незначительных изменений, после чего быстро запускается новый сайт. Проблемы начинаются, когда сайт требует не только простых изменений, и вы застреваете, меняя слабую тему. По этой причине, даже если клиенту нравится вид какой-то темы, в долгосрочной перспективе может оказаться проще создать похожую на вид тему с нуля, взяв за точку отправки стартовую тему.

Создание своей темы. Начало

Создание собственной темы будет настолько простым или сложным, насколько вы захотите. Иногда вам нужно изменить только логотип или цвет, это простой процесс. Нередко тема создается с нуля, чтобы достичь соответствия определенным требованиям или условиям или же чтобы получить нужный дизайн. Какой бы ни была ваша мотивация, в этом разделе обсуждаются основы, необходимые для быстрого создания новой темы и дизайна сайта на основе темы Twenty Eleven. В примере используется Twenty Eleven, поскольку она поставляется с WordPress, а не потому, что она считается лучшим выбором стартовой темы. На практике, однако, она используется и как стартовая, и как родительская тема для клиентских проектов.

В следующих разделах тема Twenty Eleven будет рассмотрена как пример совокупности элементов работающей темы. У вас также будет возможность изменить тему по-своему, оставив оригинальную существующую Twenty Eleven нетронутой. Следует также отметить, что в теме Twenty Eleven используются элементы HTML5 и HTML5 Shiv. HTML5 рассматривается более подробно в главе 12, но имейте в виду, что, несмотря на то что его использование с каждым днем становится все более распространенным, он по-прежнему на передовой веб-технологий и не все браузеры его поддерживают.

Основной файл: Style.css

Файл `style.css` WordPress использует, чтобы обращаться к теме. Он необходим, чтобы тема работала. Вы можете создать новую тему, ограничившись листом стиля и шаблонным файлом `index.php`, хотя этот файл может быть и пустым. Используя мощь иерархии темы, WordPress автоматически замещает недостающие шаблоны, если в теме их нет. Мы поговорим об этом позже, но уясните, что индивидуальный лист стиля — это то, что позволяет вам начать создание темы.

ЗАМЕЧАНИЕ

На деле файл `style.css` — это все, что необходимо для создания темы. См. раздел «Иерархия тем и дочерние темы» далее.

При создании `styles.css` для новой темы первые несколько строк являются критичными. Они предоставляют WordPress информацию для дальнейшего использования в консоли темы и последующего обращения к теме в ядре. Первые строки должны выглядеть так (разумеется, внесите свою информацию):

```
/*
THEME NAME: MyTheme
THEME URI: http://www.mirmillo.com/mytheme/
DESCRIPTION: Theme for my new site.  Based on Twenty Eleven.
VERSION: 1.0
```

```
AUTHOR: David Damstra (and friends)
AUTHOR URI: http://mirmillo.com/author/ddamstra
License: GNU General Public License v2 or later
License URI: http://www.gnu.org/licenses/gpl-2.0.html
Tags: dark, light, white, black, gray, one-column, two-columns,
left-sidebar, right-sidebar, fixed-width, flexible-width,
custom-background, custom-colors, custom-header, custom-menu,
editor-style, featured-image-header, featured-images,
full-width-template, microformats, post-formats,
rtl-language-support, sticky-post, theme-options, translation-ready
*/
```

Информация говорит сама за себя. Есть и дополнительное необязательное поле для иерархии темы, речь о котором пойдет далее. Убедитесь, что ваше имя уникально для вашей копии. Если вы собираетесь выпустить тему для публичного использования, бесплатно или за деньги, нужно попробовать создать уникальное имя, чтобы свести к минимуму проблемы с названиями в директории и других копиях. Кроме того, если ваша тема происходит из другой темы (конечно, по лицензионному разрешению), вам следует сохранить информацию по лицензии и авторским правам на оригинальную тему. После того как вы адресовали эту необходимую информацию WordPress, оставшаяся часть файла традиционно является CSS и предметом для соответствующих правил и структур.

Повторимся: мы делаем копию темы Twenty Eleven, превращая ее в свою собственную. Это не процесс создания дочерней темы. Функциональность дочерней темы будет рассмотрена позднее. В некоторых случаях рабочий процесс идет лучше, если новая тема создается путем копирования стартовой темы в новую папку и ее переименования и пересмотра `style.css` на предмет адекватного отображения нового проекта. Есть «за» и «против» этой техники, но для некоторых задач она работает хорошо, поскольку основная тема меняется не слишком часто, что гарантирует более сложные методологии. Кроме того, имея в работе тему, вы не захотите менять родительскую тему, вызывая каскад случайных ошибок на успешно работающем сайте. Создание копии работающей темы в новой директории устраняет зависимость от будущего тестирования в браузере, являющегося процедурой с большими затратами времени и человеческих ресурсов (ее еще не автоматизировали). В случае если в родительскую тему были внесены существенные изменения, изменения могут быть перенесены в производные темы шаг за шагом и проверены как необходимо. Создание копии стартовой темы также позволяет вам делать вручную файл CSS, меняя файлы актуальной темы, а не перезаписывая стили с добавлением дополнительных байтов багажа.

Идем дальше. Правила CSS записаны в файле `style.css`, чтобы вы могли превратить минималистический вид в профессионально спроектированную тему, которую вы создаете. Поскольку вы работаете над собственной копией Twenty Eleven, так сказать, ее ветвью, вы можете редактировать тему непосредственно в листе стиля. Кодирование CSS лежит за пределами темы этой книги. Хорошо подготовленный CSS-код представляет собой и искусство, и мастерство. Повторяем, у Wrox есть несколько книг по работе с CSS.

Показываем контент: Index.php

Создавая собственную тему, вы сталкиваетесь с проблемой курицы и яйца. Возможно, вы счастливый обладатель информации о том, как именно контент будет публиковаться на сайте WordPress и как именно он будет структурирован. Возможно, вы даже знаете, как будет выглядеть тема по окончании работы, или же профессиональный дизайнер сделал для вас несколько макетов. Но странность в том, что сайт растет естественным образом, и чтобы понять, как «выстрелит» дизайн и, соответственно, получить список стилей, нужен контент для отображения.

Вы можете использовать банки файлов контента, чтобы импортировать его на сайт и прогнать через все стили, или же можете начать строить свой сайт. Темы проверки блоков и банков файлов контента рассматриваются в главе 3.

Файл `index.php` — шаблон вашего сайта по умолчанию. У WordPress есть встроенный движок принятия решений, который определяет, какой тип информации запрошен пользователем, а после этого решает, есть ли для этого типа файл шаблона. Данная иерархия рассматривается далее в этой главе, но шаблон `index.php` является предустановленным или же вашей последней надеждой. Если WordPress не находит специального шаблона, он берет `index.php`.

Обычно файл `index.php` содержит стандартный цикл. Это традиционный формат блога, в котором записи отображаются в обратном хронологическом порядке. Например, вот фрагмент кода из файла Twenty Eleven:

```
<?php if ( have_posts() ) : ?>
  <?php twentyeleven_content_nav( 'nav-above' ); ?>
  <?php /* начинается цикл */ ?>
  <?php while ( have_posts() ) : the_post(); ?>
    <?php get_template_part( 'content', get_post_format() ); ?>
  <?php endwhile; ?>
  <?php twentyeleven_content_nav( 'nav-below' ); ?>
<?php else : ?>
```

Как уже говорилось в главе 5, цикл — это сердце WordPress. Эту концепцию крайне важно понять, поскольку именно таким образом контент выбирается и сортируется для публикации. Вы можете посмотреть на приведенный ранее фрагмент кода и увидеть, что цикл — лишь небольшая его часть и что контент записей даже не генерируется в HTML.

И вы будете правы. Предыдущий пример использует функцию `get_template_part()`, появившуюся в WordPress 3.0. Это функция WordPress, схожая с PHP-функциями `include()` и `require()`, но она более мощная, поскольку заточена под WordPress. Функция `get_template_part()` позволяет разработчику темы выудить определенные компоненты кода для повторного использования или для замены в дочерних темах.

В предыдущем примере WordPress ищет определенный PHP-файл для генерации записей в HTML. Это сравнительно сложный пример, поскольку в нем функция `get_post_format()` используется для заполнения второго параметра

включения. `get_post_format()` будет делать именно то, о чем говорит ее название, — возвращать формат записи. Форматы записи будут рассмотрены в этой главе несколько позднее. Для пользы дела нужно знать, что если запись помечена как картинка, функция вернет `image`, а если она опубликована как обычная запись или помечена как стандартная на панели добавления новой записи, функция вернет `false`.

Функция `get_template_part()` будет искать шаблон файла в директории текущей темы сначала как особую версию, используя второй параметр, а затем для первой версии, игнорируя второй параметр. Вот что будет, если вы используете следующий код в вашей теме:

```
<?php get_template_part( 'content', 'index' ); ?>
```

Сначала WordPress ищет файл `content-index.php` в папке темы и, если не находит его, переключается на `content.php`. Если вы создаете дочернюю тему и ни один из названных ранее файлов не найден, WordPress продолжит поиск в папке родительской темы.

В примере Twenty Eleven функция `get_post_format()` работает как переключатель для извлечения нужного шаблона контента, соответствующего формату записи. В теме Twenty Eleven вы заметите, что шаблонов контента много — по одному на каждый тип формата записи. Это позволяет разработчику темы контролировать, как различные форматы записи генерируются в браузере.

Другим преимуществом этой тактики является разбиение кода на меньшие фрагменты, которыми проще управлять. Вы увидите, что разбиение на фрагменты облегчает отладку, если у вас возникнут проблемы с изменениями в будущем или потребность в новой функциональности.

Отображение контента различными способами: index.php

Файл `index.php` действительно самый важный файл шаблона в теме. Хотя активная тема не может существовать в WordPress без `styles.css`, поскольку только так WordPress узнает, что тема есть, — `index.php` осуществляет тяжелую работу.

На заре WordPress шаблон `index` был единственным шаблоном. Вся тема была одним этим файлом, представлявшим собой просто-напросто цикл. Это прекрасно работает для WordPress, если вы используете традиционный блог и такой блогообразный вид, вероятно, является причиной, по которой WordPress по-прежнему хотят использовать как движок для блога.

Хочется надеяться, что вы читаете эту книгу, поскольку знаете, что WordPress может гораздо больше, или же, если не знаете, то понимаете это сейчас. Шаблон `index` крайне важен, его нельзя переоценить. Это шаблон, который WordPress будет использовать как последний вариант, если не сможет найти что-либо более специфическое (см. раздел «Иерархия шаблонов» далее в этой главе).

Так или иначе, файл `index.php` не должен быть единственным циклом, показывающим самые последние записи. Это весьма традиционный вариант, который, возможно, хорошо подойдет для вашего сайта, но вы можете пойти и другим путем. Файл `index.php` может быть структурирован самыми разными способами, их количество безгранично. Он может содержать множество различных циклов для различных меток или категорий или не содержать ни одного. Шаблон `index` может функционировать как страница ошибок, если у вас достаточно специфических шаблонов для всех других фрагментов контента вашего сайта.

Создание своей темы: DRY

Как говорилась выше, вот что является основами. WordPress требует файл `style.css` с правильно отформатированной информацией в заголовке; в теме должен быть шаблон `index.php`. Теперь нужно расширить тему, чтобы использовать больше файлов шаблонов и движок темы в WordPress.

Хороший разработчик знает, что не стоит повторять код в разных местах; это скверный дизайн «с душком». (Вы сами это знаете!) Такой дизайн получил название DRY (Don't Repeat Yourself, «Не повторяйся») и от него совсем не сложно избавиться. Как только вы испытываете искушение вырезать фрагмент кода из одного шаблона и вставить его в другой, «душок» пошел. Вот как вы можете разбить шаблоны на части для повторного использования. Есть три очевидных компонента, использующихся практически на всех страницах сайта, чтобы он выглядел единообразно и структурированно. Заголовок, «подвал» и информация на боковых панелях обычно одинаковы для всех страниц. Вы научитесь расщеплять эти файлы с применением дополнительной логики, чтобы иметь дело с исключениями в дизайне.

header.php

Вам может показаться, что файл назван так ошибочно, но это стандартное имя данного файла, которое ищет WordPress. Файл `header.php` включает в себя все выше генерируемой страницы до области контента. Это может вызвать удивление, поскольку правильно сформатированный документ HTML включает в себя собственную информацию `<head>` со своими особыми требованиями. Файл шаблона `header.php` включает в себя заголовок HTML, но также и начало документа HTML, а обычно еще и логотип и навигацию сайта, если вы используете горизонтальную схему навигации по верху. Он также может включать в себя любые дополнительные элементы, такие как вторичная навигация или область поиска.

Поскольку данный файл куда больше, чем заголовок HTML, есть тенденция использовать термин, пришедший из печати, и называть эту область *nameplate*, по аналогии с шапкой газеты или журнала. Однако мы будем придерживаться традиций и сохраним имя файла как `header.php`, чтобы не нарушать совместимость со встроенной функциональностью WordPress.

Создавая файл шаблона заголовка, важно включить функцию WordPress `wp_head()`. Это зацепка, позволяющая WordPress прикреплять определенную функциональность к заголовку сайта, она также используется плагинами.

Функция `wp_head()` включается в HTML-узел `<head>` и является критически важной для долгосрочной совместимости и функциональности темы. Так что убедитесь, что она на месте.

Когда вы выделили раздел «шапки» страниц в отдельный шаблон `header.php`, то, если вы создавали тему с нуля, вам нужно скорректировать файл `index.php` таким образом, чтобы в него вошел этот раздел. Вы можете использовать традиционное семейство функций PHP `include` или `require`, но у ядра WordPress есть и своя функция для тем. Она схожа с функцией `get_template_part()`, о которой мы говорили ранее, но является специфической функцией для шаблона заголовка. Просто добавьте следующий код в верхнюю часть файла `index.php` (и подчиненные файлы, о которых речь пойдет далее):

```
<?php
    get_header()
?>
```

Эта функция автоматически включает имя файла `header.php` из директории текущей темы в текущий файл для генерирования. У функции нет другой дополнительной функциональности сверх PHP `include`, кроме определения правильного пути `include`, но она куда более понятна при работе над темой.

При желании вы можете выделить дополнительные компоненты из файла `header.php` и включить их обратно с помощью PHP `includes`. Иногда, если у сайта особенно длинная или сложная глобальная навигация, она разбивается на несколько шагов для включений. На практике работать с файлами меньшего размера проще в отношении редактирования, поскольку у каждого файла шаблона своя особая функция и сложность отладки снижается.

ЗАМЕЧАНИЕ

В прошлой жизни один из авторов был призван работать над веб-приложением, которое было создано как один файл, а функциональность осуществлялась посредством запуска особых функций. Хотя функции были аккуратно разделены, каждый раз при необходимости отладки приложения сообщения об ошибке были практически бессмысленными. Хотя номер строки и менялся (одна из тысяч сигнальных ракет), все они находились в файле `index.php`, и для определения того, что случилось, неизменно приходилось просматривать все приложение.

Представьте, насколько проще было бы устранить проблему в приложении, если бы в сообщении об ошибке указывалось, что ошибка произошла в 100-й строке рассматриваемого файла, а не в 10 000-й строке файла всего приложения.

Каждый когда-либо писал плохой код, и конечно, мы (авторы) не исключение, хотя и не делали этого умышленно. Все, что мы говорим: доставьте себе удовольствие и разбейте код на небольшие, удобно управляемые файлы, если это возможно.

footer.php

Так же как и в случае с файлом `header.php`, все ниже области контента должно быть выделено в отдельный «подвал». Природа этих файлов недавно изменилась. Исторически они содержали информацию по авторским правам и контактную информацию, но в последние годы эта часть «недвижимости» была расширена для включения навигационных параметров и информации, относящейся к сайту. Самые современные темы, в том числе Twenty Eleven, содержат в «подвале» области виджетов для произвольного контента. Области виджетов мы обсудим в этой главе далее. Что именно помещать в «подвал», решать вам, но поскольку он по большей части остается одним и тем же на всех страницах, это первый кандидат на выделение в отдельное включение.

Не забудьте убедиться, что вы включили функцию `wp_foot()` в шаблон «подвала». Это функция позволяет WordPress включать любую необходимую информацию из активированных плагинов и, как правило, включает закрывающие теги `</body></html>`.

Как и в случае включения шаблона заголовка, WordPress предлагает схожую функциональность для информации «подвала» с отдельной специальной функцией включения. Добавьте следующий код в нижнюю часть файлов шаблона:

```
<?php
    get_footer()
?>
```

sidebar.php

Еще один кандидат для выделения — боковая панель, то есть все, что находится слева или справа от контента. Боковая панель может содержать навигацию по сайту, если выбрана вертикальная схема навигации, или менее важную дополнительную информацию по контенту.

Тема Twenty Eleven позволяет делать левую или правую боковую панель, используя один файл `sidebar.php` и помещая его с CSS. Однако у вас может быть много файлов боковых панелей для каждой колонки или специфических боковых панелей на разных страницах.

При работе с боковыми панелями нужно иметь в виду ряд различных проблем. Сначала необходимо решить, сколько боковых панелей вам нужно, затем — будут они статичными, с виджетами или гибридными. Наконец, следует определить, как структурирован HTML, чтобы CSS мог помещать боковые панели в нужные места. Затем нужно провести проверку во всех целевых браузерах. Такова жизнь веб-разработчика.

Как уже было отмечено, боковые панели Twenty Eleven представляют собой один файл, неважно, правая панель или левая, и полностью виджетированы. Это позволяет делать заготовку файла с относительной легкостью и использовать консоль WordPress для размещения виджетов по необходимости.

В файлы шаблонов нужно поместить следующий код, чтобы добавить файл `sidebar.php`:

```
<?php get_sidebar(); ?>
```

Иногда наличие обеих боковых панелей в одном файле неудобно с позиций дизайна или, скорее, CSS. Или же вы выделили панели в отдельные файлы. В любом случае, вы можете создать две боковые панели для традиционного размещения слева и справа. Например:

```
<?php
    get_sidebar('right');
?>
```

берет файл `sidebar-right.php`, как видно в параметрах вызова функции, и помещает его в нужное место. Эта функция опять похожа на функцию `get_template_part()`, но сделана специально для боковых панелей и куда более читаема в коде.

У продвинутых каркасов тем много боковых панелей, которые отличаются от привычных представлений о том, что боковая панель — это вертикальное пространство справа и слева от контента. У некоторых из таких тем есть дополнительные панели над, под и в центре циклов записи. Наличие нескольких виджетированных областей такого рода наделяет администратора дополнительными возможностями, поскольку теперь он может размещать виджеты WordPress по всей странице.

При работе с боковыми панелями важно соблюдать баланс между виджетированными зонами, — подразумевая то, что они могут контролироваться и управляться создателем контента в консоли, — и жестко кодированными зонами в PHP-файле шаблона. Виджеты могут быть очень мощными, особенно благодаря многочисленным плагинам. Но бывает и так, что работу выполняет PHP-код в файле шаблона, администратору не нужно обновлять контент или же встроенная функциональность WordPress самообновляется. Баланс — выбор разработчика.

Отклонения от нормы: условные теги

Вы были хорошим разработчиком и выделили повторяющиеся фрагменты кода в отдельные шаблоны или файлы-включения. Хорошая работа. Но только что позвонил директор по маркетингу, и хотя сайт уже почти готов, а дизайн был им подписан, он забыл сказать вам, что на всех страницах и во всех записях категории «Пони» в шапке рядом с логотипом должна быть миленькая радуга. Оставим в стороне вопросы вкуса, проблема в том, что у вас один общий файл `header.php`, а теперь только некоторые из заголовков нужно изменить особым образом.

Как и в случае со всеми вещами с открытым кодом, есть много способов разрешить эту ситуацию. Вы можете разобраться с таким простым примером с помощью всего лишь ловко слаженного CSS и класса темы body. Или же можно создать целый файл шаблона категории (см. далее), чтобы изменить ее стиль. Но поскольку вы

имеете дело всего лишь с крошечным элементом, создавать для него новый шаблон — явный перебор.

Но погодите: не все потеряно. Разработчики WordPress уже имели дело с директорами по маркетингу и знакомы с такими ситуациями, которые происходят постоянно, вот почему они включают условные теги. У WordPress есть множество встроенных условных тегов, рассматривать каждый не входит в задачи этой книги, не говоря уже о том, что это невыносимо скучно. Но эти теги существуют и могут сослужить хорошую службу, например в тех случаях, когда нужно определить, какой тип страницы или какая метаинформация о контенте будет отображаться.

Чтобы утихомирить директора по маркетингу, вам нужно включить в файл `header.php` нечто подобное:

```
<header id="branding" role="banner">
  <hgroup>
    <h1 id="site-title"><span><a
      href="<?php echo esc_url( home_url( '/' ) ); ?>"
      title="<?php echo esc_attr( get_bloginfo( 'name', 'display' ) ); ?>"
      rel="home"><?php bloginfo( 'name' ); ?></a></span>
    </h1>
    <?php
      if (is_category('Ponies')) { ?>
        // если это рубрика 'Ponies', то наложим на логотип разноцветную радугу
        <img id="raibow"
          src="<?php bloginfo('template_directory');?>/img/rainbow.png"
          alt="OMG! Ponies! " />
      <?php } ?>
      <h2 id="site-description"><?php bloginfo( 'description' ); ?></h2>
  </hgroup>
  ...
</header>
```

Теперь всякий раз, когда категория контента текущей страницы оказывается «Пони», заголовок также будет включать в себя `rainbow.png`. С альфа-прозрачностью PNG все будет выглядеть прилично. Данный пример работает только для категории страниц, а не для страниц отдельных записей в категории «Пони».

Создание своей темы: отображение контента

Хорошая тема улучшает контент сайта. Речь идет не только о внешнем виде, пригодности для данного сайта и соответствии бренду. Тема также должна верно структурировать контент. В WordPress есть множество разных шаблонов и функциональность, соответствующая потребностям любого типа сайта. Проблема в том, чтобы найти лучшую комбинацию файлов шаблонов для создания оптимального устройства контента. Не во всех темах есть все файлы шаблонов, в большинстве их нет. Так что лучше скомбинировать и дополнить подходящие.

Индивидуализация домашней страницы: front-page.php

Домашняя страница — кто-то еще использует этот термин? Он как будто бы пришел из 1990-х, но как еще ее назвать? В этом разделе рассматривается первая страница сайта, отображаемая при переходе по корневому URL. Пользователи Apache знают, что index-страница сайта — это домашняя страница, или просто index, на сервере Microsoft IIS она называется default. В консоли WordPress — это front page, так что можете называть ее и так.

У темы всегда должен быть файл `index.php`, поскольку это последний шаблон, который будет применяться, когда все остальное не подошло. Что, если вы хотите, чтобы у титульной страницы была особая разметка, возможно, содержащая какую-то информацию о сайте — например, страницы продуктов? Вам вряд ли захочется связываться с `index.php`, поскольку придется переделывать всю тему, и все это только для специальной разметки в одном месте.

Для реализации этого сценария есть плагины и другие уловки. Вы даже можете использовать консоль WordPress, чтобы установить статичную титульную страницу, являющуюся одной из опубликованных страниц сайта. Но в данном случае мы говорим о генерировании произвольной разметки или HTML, а не о традиционной странице. Самый простой способ — использовать встроенную иерархию шаблона и задать особую титульную страницу с помощью шаблона `front-page.php`.

На самом деле есть два файла шаблонов, которые могут функционировать как титульная страница: `front-page.php` или `home.php`. В некоторых более старых темах и даже в первом издании этой книги единственным вариантом был `home.php`. С появлением WordPress 3.0 `front-page.php` стала самым предпочитаемым именем файла шаблона для титульной страницы. Сюда вовлекается несколько больше, в зависимости от того, какие параметры чтения вы установили в консоли. Далее в этой главе вы увидите иерархию шаблонов и то, как два шаблона соотносятся между собой.

Создание особой разметки и, таким образом, файла шаблона для титульной страницы полезно, если титульная страница уникальна. Все больше и больше создание уникальной титульной страницы становится маркетинговым инструментом. Вот некоторые причины этого:

- Демонстрация особенностей товара или услуги.
- Демонстрация особенностей других разделов веб-сайта.
- Перенаправление трафика в определенный раздел сайта.
- Пошаговое объяснение процессов, связанных с товаром.
- Охват уровней предоставляемой услуги.

Посмотрите базовый пример на рис. 9.1, где на титульной странице демонстрируются товары или услуги, предлагаемые на веб-сайте. У них есть собственные

страницы или записи на сайте. На титульной странице есть симпатичные демонстрационные поля со ссылками на индивидуальные страницы. Для усовершенствования этих полей и прокрутки в них изображений используется jQuery. Или же вы можете использовать различный инструментарий JavaScript toolkit или Adobe Flash, но jQuery уже включен в WordPress и, честно говоря, он крут. Так почему бы его не использовать? Нижняя часть раскладки включает в себя раздел последних новостей.

Если вы еще не догадались, все это потребует привлечения нескольких циклов. Первый цикл будет создавать контент для демонстрационного поля. Он будет извлекать записи из определенной категории или записи индивидуального типа. Таким образом, администратор сайта при необходимости сможет добавлять и убирать контент из демонстрационного поля, совсем не заглядывая в код. Конечно, будут определенные ограничения по дизайну, такие как размер изображений и формат, и, возможно, определенные правила, которым нужно будет следовать в записях. Но способность менять эту информацию в консоли WordPress — мощный инструмент.

Цикл демонстрационного поля (*showcase loop*), иногда именуемый *слайд-шоу* (*slideshows*), может выглядеть примерно следующим образом (в данном случае используются записи индивидуального типа, о которых говорилось в главе 7):

```
<div id="showcase">
  <?php
  global $post;
  $args = array(
    'post_type' =>'slides',
    'numberposts' => -1,
    'orderby' => 'rand'
    );
  $slider_posts = get_posts($args);
  // отображаем демонстрационные поля, только если существуют слайды
  if($slider_posts) {
    foreach($slider_posts as $post) : setup_postdata($post);
      // получаем изображение
      $thumbnail = wp_get_attachment_image_src(get_post_thumbnail_id(),
        'home-slide');
      if ($thumbnail[1] == "600" && $thumbnail[2] == "160") {
        // проверяем размеры миниатюры в css ?>
        <div id="feature-<?php echo $post->ID; ?>" class="slide">
          <a href="<?php the_permalink(); ?>" title="<?php the_title(); ?>">
          <img src="<?php echo $thumbnail[0]; ?>"  title="<?php the_title(); ?>" />
          </a>
        </div>
      <?php  } ?>
    <?php endforeach; ?>
    <?php wp_reset_postdata(); ?>
  <?php } ?>
</div>
```

Так создается генерируемый HTML, как показано на рис. 9.1.

Давайте разберемся, что здесь происходит. Весь цикл демонстрационного поля заключен в `<div>` с `ID` или `showcase`. Это чтобы jQuery мог позднее «зацепиться» за него. В коде PHP создается произвольный запрос для цикла. Запрос ищет записи индивидуального типа, называемые слайдами, которые вы ранее утвердили в файле `functions.php`, а также задали в консоли WordPress. Этот запрос выбирает слайды из записей индивидуального типа и возвращает их в произвольном порядке. После этого цикл приступает к созданию элементов `<div>`, у каждого из которых есть уникальный `ID`, опять-таки для jQuery через CSS. Пользовательский тип записей задается для использования поддерживаемых WordPress изображений как контента слайдов. Это позволяет администратору сайта загружать изображения слайдов, в свою очередь ведущие на страницы с более подробной информацией в теле записи. Предыдущий пример кода отобразит в поле графику в слайде, только если она соответствует определенным размерам: 600 px ширины при высоте 160 px. Наконец, доступны бесчисленные плагины jQuery, которые могут превратить этот пока неясный блок контента на странице в элегантное слайд-шоу.

Нижняя секция может быть традиционным циклом, схожим с циклом шаблона `index.php`. Тема Twenty Eleven не поставляется со встроенными шаблонами титульной или домашней страницы, поскольку при использовании этих шаблонов вы уже оказываетесь на пути создания произвольной темы. Также это не единственный способ получить данную функциональность. Как и в WordPress 2.1, вы можете контролировать, что отображается на титульной странице, и определять ее как созданную вами статическую страницу, после чего создавать особый шаблон страницы для завершения описанных ранее дизайнерских решений. Вы также можете строить несколько циклов — один для использования записей категории для слайдов, другой — для исключения категории из новостей. Это уж как вы привыкли. Еще раз: выбор внешнего вида — одно из тех решений, которое вам необходимо сделать как разработчику, находя баланс между потребностями клиента и легкостью поддержания сайта.

Отображение старых записей: archive.php

Если вы человек прилежный, у вашего сайта в конце концов появится старый контент. А если вы искусны, то сможете развлекаться, создавая записи типа: «Год назад на моем сайте я рассказывал вам о...» Наконец, содержимого может оказаться столько, что его будет уже невозможно отобразить или вызвать для отображения на титульной странице. То есть если контент генерировался регулярно, рано или поздно наступит момент, когда вам захочется сослаться на что-то, чего больше нет на титульной странице или в списке последних записей. Значит, пришло время нырнуть в подземелье прошлого содержимого.

Тут в дело вступает шаблон `archive.php`. Есть множество вариантов представления старого контента. Если вернуться к блог-корням WordPress, то самым очевидным методом будет продолжать обратный хронологический порядок.

Рис. 9.1. Файл особого шаблона может придать титульной странице уникальный вид

Если шаблона архива нет, WordPress просто использует шаблон index для отображения старых записей. В теме Twenty Eleven есть интересный шаблон `archive.php`, ведущий начало от оригинальных стартовых тем, таких как Sandbox. Это крайне гибкий подход, создающий визуальное отображение архива на базе даты. Рассмотрим код шаблона `archive.php` из Twenty Eleven:

```
<h1 class="page-title">
  <?php if ( is_day() ) : ?>
    <?php printf( __( 'Daily Archives: %s', 'twentyeleven' ), '<span>' .
      get_the_date() . '</span>' ); ?>
  <?php elseif ( is_month() ) : ?>
    <?php printf( __( 'Monthly Archives: %s', 'twentyeleven' ), '<span>' .
      get_the_date( _x( 'F Y', 'monthly archives date format',
      'twentyeleven' ) ) . '</span>' ); ?>
  <?php elseif ( is_year() ) : ?>
    <?php printf( __( 'Yearly Archives: %s', 'twentyeleven' ),
      '<span>' . get_the_date( _x( 'Y', 'yearly archives date format',
      'twentyeleven' ) ) . '</span>' ); ?>
  <?php else : ?>
    <?php _e( 'Blog Archives', 'twentyeleven' ); ?>
  <?php endif; ?>
</h1>
```

Этот блок кода показывает, как шаблон архива темы отображает уникальный заголовок в зависимости от того, ищет посетитель архивные записи за день, месяц или целый год или отслеживает условное разбиение на страницы.

За исключением того факта, что Twenty Eleven изначально основана на датах, особый шаблон архива совершенно не важен. Хотя иметь информацию по дате полезно при определении степени свежести информации. Обращаетесь ли вы на самом деле к записям, опубликованным в мае 2007-го? Скорее всего, вы ищете записи на определенную тему или относящиеся к определенной категории.

Отображение одной рубрики: category.php

Зайдем в шаблон категории. Шаблон `category.php` создает цикл записей одной определенной категории. Шаблон категории запускается, когда посетитель выбирает определенный URL с именем категории. Например: http://example.com/category/zombies. В шаблоне `category.php` WordPress уже определил, что пользователь ищет записи в определенных запросах категории, поэтому предустановленный цикл автоматически делает для вас запрос, не требуя специального взаимодействия.

Используя этот шаблон, вы можете в целом отобразить записи и информацию по категории. Именно так устроена тема Twenty Eleven. Например, тема Twenty Eleven помещает заголовок и информацию по категории, полученную из ее описания, если оно есть:

```
<header class="page-header">
  <h1 class="page-title"><?php
    printf( __( 'Category Archives: %s', 'twentyeleven' ), '<span>' .
      single_cat_title( '', false ) . '</span>' );
    ?>
  </h1>
  <?php
  $category_description = category_description();
  if ( ! empty( $category_description ) )
    echo apply_filters( 'category_archive_meta',
    '<div class="category-archive-meta">' . $category_description . '</div>'     );
  ?>
</header>
```

Здесь рассматривается случай предустановленной рубрики, то есть предустановленный шаблон, который хорошо бы иметь. Но что, если вы хотите, чтобы у каждого шаблона рубрики был уникальный вид — например, цветовая схема или иконка?

Предположим, что очарованный пони и радугой директор по маркетингу теперь хочет рубрику Zombie. Вместо того чтобы использовать условные теги, вы можете создать особый шаблон рубрики. В соответствии с иерархией шаблонов WordPress будет искать, существует ли шаблон рубрики для рубрики, запрошенной в URL. Если вы еще не заметили этого: WordPress работает от частного к общему, пока не найдет подходящий шаблон. WordPress пытается выбрать шаблон, наиболее точно заточенный под тип запрошенной информации, и работает далее, переходя ко все более общим шаблонам, пока не доберется до предустановленного `index.php`. Это критически важно знать при определении шаблонов темы, и мы к этому еще вернемся.

Для директора по маркетингу вы можете создать, например, шаблон `category-3.php`, поскольку у рубрики Zombies ID 3.

Самый простой способ найти номер ID рубрики — навести мышь на имя рубрики во вкладке **Редактировать рубрику** в консоли и увидеть его на панели статуса в нижней части окна браузера, как показано на рис. 9.2.

Это отчасти проблема курицы и яйца, если вы хотите создать шаблон для определенной рубрики. Чтобы назвать файл шаблона правильно, нужно сначала создать рубрику, чтобы получить ее ID.

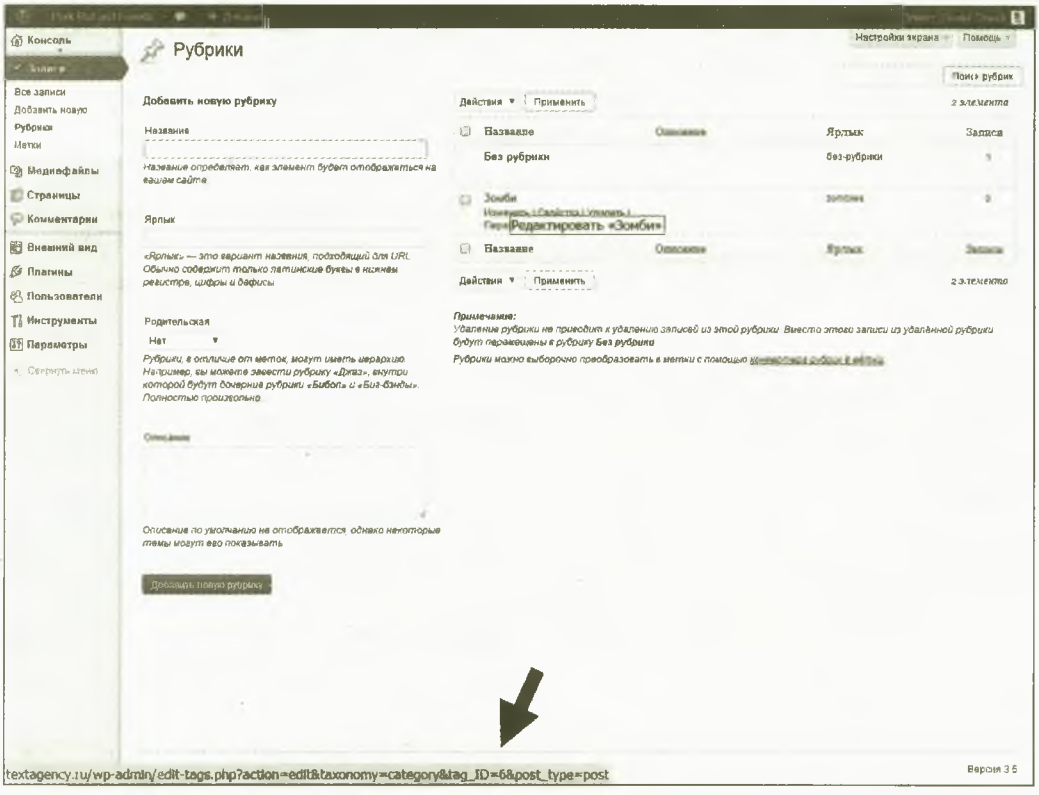


Рис. 9.2. Наведение мыши на имя рубрики в консоли для получения ID в панели состояния

К счастью, есть и другой путь. Начиная с WordPress 2.9 пользователи могут создавать шаблоны рубрик, использующие слаги. А файлу с шаблоном со слагом WordPress отдает предпочтение перед основанным на ID. Чтобы избежать проблемы курицы и яйца, можно сначала создать рубрику и снабдить ее слагом, в данном случае `zombies`. Сделав это вовремя, вы можете создать файл шаблона рубрики `category-zombies.php` и отдыхать.

Эти специфические шаблоны рубрик работают точно так же, как и общие шаблоны рубрик, и выбирают записи из рубрики автоматически. Технически это работает

несколько иначе: WordPress уже знает, какие записи будет отображать, и определяет просто, каким образом их показывать. С использованием специфических шаблонов рубрик вы добиваетесь гибкости в стиле для каждой рубрики.

Возможно, вы уже заметили, что в WordPress всегда есть более одного варианта какого-либо действия. Данный простой пример с директором по маркетингу можно решить с помощью условных тегов или особых для рубрики шаблонов, или же, что наиболее вероятно, вы можете удовлетворить его требования, используя CSS, поскольку в большинстве тем есть множество ловушек CSS. Расширяемость этой возможности — решающий аспект. Знание того, что в WordPress есть встроенные возможности, когда-нибудь вас спасет.

Отображение записей по метке: tag.php

Шаблон `tag.php` функционирует примерно так же, как шаблон `category.php`. Он запускается, когда посетитель запрашивает определенную метку. Имеет смысл, только если вы активно снабжаете метками контент сайта. Скорее всего, вы задаете метку для содержимого рубрики, поскольку такой способ организации кажется более естественным, но смысл снабжения метками не всегда настолько очевиден и нередко считается чем-то дополнительным.

Так или иначе, если вы искушены в снабжении метками, такой шаблон будет приятным бонусом для вашей разметки и может быть полезен для пересекающихся записей со схожим содержимым. При вызове данного шаблона цикл автоматически заполняется записями с определенной меткой для генерирования. Чтобы подробнее рассмотреть работу цикла, загляните в главу 5.

Точно так же можно создавать шаблон для определенной метки. Как и в случае с рубриками, вы можете использовать для создания шаблона ID метки или слаг. Если вам нужен отдельный шаблон для метки Zombies, используйте слаг метки, чтобы создать `tag-zombies.php`. Вам нужно подтвердить слаг метки во вкладке Управление метками в консоли.

Использование шаблонов рубрики и метки может и не быть тем способом отображения контента, который вам видится, особенно если вы используете WordPress как систему управления содержимым. Однако просто включение этих шаблонов добавит функциональности и настраиваемости ядру WordPress. Эти шаблоны позволят посетителям исследовать сайт разными способами и, возможно, привлекут к нему немного больше внимания, поскольку содержимое можно будет рассматривать по-новому. Рубрики и метки группируют схожий контент. Использование таких шаблонов создаст органичные возможности изучения всего сайта.

Не отмахивайтесь от этих шаблонов, оставляя их простым списком по обратной хронологии, как в случае со страницей архива. Придумывайте креативные варианты презентации контента. Раз уж у вас есть пользователи, заинтересовавшиеся определенным содержимым, почему бы не привлечь их внимание к схожей информации?

Другие архивные шаблоны

С учетом всего вышесказанного архивные шаблоны WordPress буквально расцвели в последнее время. Помимо шаблонов, о которых речь шла ранее, вы можете создать особые архивные шаблоны для произвольных таксономий или индивидуальных типов записей. Хотя эти шаблоны и попадают в иерархию архивных, вы можете рассматривать их как простое группирование контента по определенному признаку.

Если в теме или для контента заданы индивидуальные типы записей, как в примере со слайдами в демонстрационном поле, рассмотренном ранее, у вас также могут быть произвольные шаблоны архивных страниц. Правила их наименования несколько отличаются. Для слайдов вы создадите файл шаблона с названием `archive-slides.php`, где *`slides`* — имя пользовательского типа записей.

Точно так же, используя в теме произвольные таксономии, вы можете создать произвольные архивные шаблоны для таксономии и особых элементов. WordPress выберет наиболее подходящий файл шаблона, так что шаблоны элементов пойдут перед более общими шаблонами таксономий. Пользовательские типы записей и таксономии рассматривались в главе 7.

Как показать отдельную запись: single.php

Вы создали приманку в виде отличной первой строки записи. Что-то остроумное и завлекательное. После поклевки — ловушка на выдержке из записи, и вы поймали посетителя. Он щелкнул, чтобы прочесть окончание статьи.

Шаблон `single.php` — наиболее вероятная площадка приземления по прибытии посетителей через поисковую систему. С учетом того, что у вас просто превосходный контент, поисковая система расположит страницы с подробностями выше страницы `index`, содержащей лишь список выдержек. Поэтому лучше всего вложить некоторое количество времени в этот шаблон, поскольку его просматривают наиболее часто. Насыщение этого шаблона тизерами записей и прочего контента на схожую тему только повысит вероятность дальнейшего исследования посетителем вашего сайта, добавления его в закладки, подписки на фиды и, лучше всего, возвращения. Все эти варианты повысят ваш уровень в поисковых системах.

Вы можете отобразить все содержимое отдельной записи с помощью файла шаблона `single.php`. WordPress решил, что пользователь запросил все содержимое отдельной записи, поэтому шаблон должен содержать не цикл, а просто вызов функции `the_post()` для получения данных и базы данных. Если вы посмотрите на тему Twenty Eleven, то заметите, что шаблон `single.php`, однако, использует цикл и функцию `get_content_part()`, чтобы поддерживать единообразие всех шаблонов, но поскольку отображается только одна запись, это чрезмерно.

Если запись слишком длинная, встроенная функциональность WordPress или особые плагины позволяют разбить ее на несколько страниц. Пользователи

испытывают по этому поводу смешанные чувства. Хотя исследования давно выявили оптимальную длину строк и объем контента, позволяющие повысить читабельность, некоторые посетители бурно протестуют по поводу времени загрузки при разбиении на страницы. Это выбор дизайна, основывающийся на типе контента и виде страницы.

Добавление ссылок, имеющих отношение к записи, — отличный способ побудить посетителей исследовать сайт далее. Несколько плагинов могут добавить близкое по теме содержимое в нижнюю часть страницы отдельной записи или просканировать контент по ключевым словам и ссылкам. На практике нужно все это попробовать, чтобы увидеть, как оно будет работать на вашем сайте.

Альтернативой для бедных является добавление простой рубрики или цикла меток для получения записей по схожим темам внизу страницы. Например:

```
<h2>Other posts in this category</h2>
<ul id="related">
  <?php
  $category = get_the_category();
  $my_query = new WP_Query("category_name=".$category[0]->name."
      &showposts=5&orderby=rand");
  while ($my_query->have_posts()) : $my_query->the_post();
    echo '<li><a href="'. $post->permalink.'">"' . $post->post_title .'"</a>
            </li>';
  endwhile;
  ?>
</ul>
```

Здесь вы получаете пять случайно выбранных записей из первой рубрики данной записи. Это не самый разумный, но простой способ отобразить некоторые ссылки на связанный контент на странице отдельной записи. Другой возможностью является отображение дополнительных записей того же автора:

```
<h2>Other posts by this author</h2>
<ul id="related">
  <?php
  $author = get_the_author_meta('id');
  $my_query = new WP_Query("author=".$author&showposts=5&orderby=rand");
  while ($my_query->have_posts()) : $my_query->the_post();
    echo '<li><a href="'. $post->permalink.'">"' . $post->post_title .'"</a>
      </li>';
  endwhile;
  ?>
</ul>
```

Хотя шаблон `single.php` подойдет для большинства сайтов, WordPress предлагает больше возможностей настройки пользовательских типов записей. Используя приведенный ранее пример демонстрационного поля со слайд-шоу, вместо генерирования страницы по щелчку с применением `single.php` можно создать особый шаблон `single-slide.php`, который может сделать поддерживаемое изображение или другие произвольные поля более притягательными или эффективными.

Отображение страницы: page.php

Когда вы используете WordPress как систему управления контентом, вам приходится принимать кое-какие решения, например использовать страницы или записи. Тут как с домашними животными: есть кошатники, а есть собачники. По большей части в этой главе речь идет о записях и об индивидуальных типах записей.

Работая с клиентом, вы обычно создаете смешанный дизайн, который использует и страницы, и записи. Записи годятся для элементов на основе времени, таких как новости и акции, тогда как страницы отображают статичную информацию, меняющуюся не слишком часто, например товары и услуги. Страницы товаров дополняются записями по схожей тематике. Это дает клиенту преимущество использования записей для привлечения трафика к статичным страницам товаров.

Шаблон `page.php` работает по сути так же, как и шаблон отдельной записи. Цикла нет — если только вы не создали особый шаблон, но технически это другой файл шаблона — только вызов `the_post()`. Да, это такая же функция, как и в `single.php`. WordPress считает записи и страницы одним и тем же типом контента, и `the_post()` выбирает его из базы данных.

Как и в предыдущих случаях, вы можете иметь особые страницы шаблонов для определенных страниц на основе ID страницы или слага. Тут схема та же, что и ранее. Кроме того, у вас могут быть индивидуальные шаблоны страниц, которые можно применять к любой странице на сайте. Но об этом мы поговорим подробнее далее.

Отображение приложений к записи: attachment.php

Честно говоря, мы (авторы), не думаем, что вы когда-либо будете использовать эти файлы шаблонов добровольно. Впервые появившийся в WordPress 2.5, `image.php` был особым шаблоном для отображения — чего бы вы думали? — изображений из галереи. С тех пор эта ветвь иерархии шаблонов выросла и обобщилась до отображения любых приложений, которые вы можете добавить к записи на основе типа MIME. У многих тем просто нет этого набора шаблонов. Однако в Twenty Eleven есть шаблон `image.php`, но не `attachment.php`. По сути, он работает весьма схоже с `single.php`, настолько, что `single.php` — следующий применяемый шаблон, если этого не существует.

Наиболее распространенный вариант использования данного шаблона — создание шаблона `image.php`. Этот шаблон обеспечивает особый шаблон только для просмотра медиагалереи. В галерее могут находиться различные типы медиа, то есть она не ограничена изображениями. Этот шаблон будет вызываться для любого медиаэлемента, если только для него не определен более специфический шаблон, обычно он включает в себя описание медиа и функциональный комментарий. Этот шаблон крайне полезен для сайта типа портфолио, например для фотостудии или художественной коллекции. Еще раз: этот шаблон функционирует практически идентично шаблону отдельной записи — с небольшими различиями из-за генерирования изображения, а не параграфа.

Схожим образом типы шаблонов могут использоваться с типами медиа, отличными от изображений. Вы можете создать шаблоны для видео, аудио или приложений. Однако это будут весьма особые случаи использования, так что в реальности вы редко будете испытывать потребность в этих шаблонах, если только вы не создаете специфический сайт.

Иерархия шаблонов

Как WordPress выбирает, что использовать, при таком разнообразии шаблонов? Ядро WordPress довольно умное в этом отношении. На основе URL WordPress определяет, какой тип контента был запрошен. Затем WordPress выясняет степень специфичности используемого шаблона, пытаясь сначала применить наиболее специфические шаблоны, соответствующие критериям, а затем переходя к более общим до нахождения соответствия. Эта система работает хорошо, будучи толерантной к ошибкам посредством постоянного отката назад к `index.php`, и при этом является невероятно мощной, потому что как разработчик вы можете при необходимости создавать произвольные шаблоны для весьма специфических ситуаций.

Это проще всего проиллюстрировать с помощью блок-схемы, показанной на рис. 9.3. Она позаимствована из Кодекса WordPress, но более полная версия доступна онлайн.

Как видите, это замечательное древо решений, гибкость которого велика. Не во всех темах есть файлы шаблонов или необходимость в них. Но, конечно, более настраиваемые или созданные для особых целей темы могут выгадать от этой иерархии и создать уникальное приложение WordPress.

Также полезно отметить, что иерархия поиска шаблонов определяется в `template-loader.php`, где зацепка задается до начала древа поиска — `template_redirect` — это позволяет вам изменить процесс выбора шаблона. Все это используется по большей части для работы с перенаправлением URL, например с «сокращенной ссылкой», определенной для страницы, которая может скрывать некоторую информацию URL, обычно используемую для дешифровки того, что использует шаблон WordPress.

ВНИМАНИЕ

На некоторых построенных нами сайтах случалось такое, что у рубрик, меток, страниц и даже авторов были одинаковые или схожие таксономии. Например, на корпоративном новостном сайте может быть страница об отделе и рубрика отдела для новостей о нем, а некоторая информация при этом вполне может быть снабжена меткой в виде названия отдела. В подобных случаях древо решений WordPress может не сработать и выдать непредсказуемый выбор. С проблемой можно справиться несколькими способами: тщательно выбрать таксономию, убедившись, что каждый слаг достаточно индивидуален, во избежание сложностей, или же принудительно задать желаемый вариант поведения через файл `.htaccess`. Здесь основной проблемой является то, как WordPress обрабатывает постоянные ссылки, сводящиеся, по сути, к схеме, соответствующей метаданным слага.

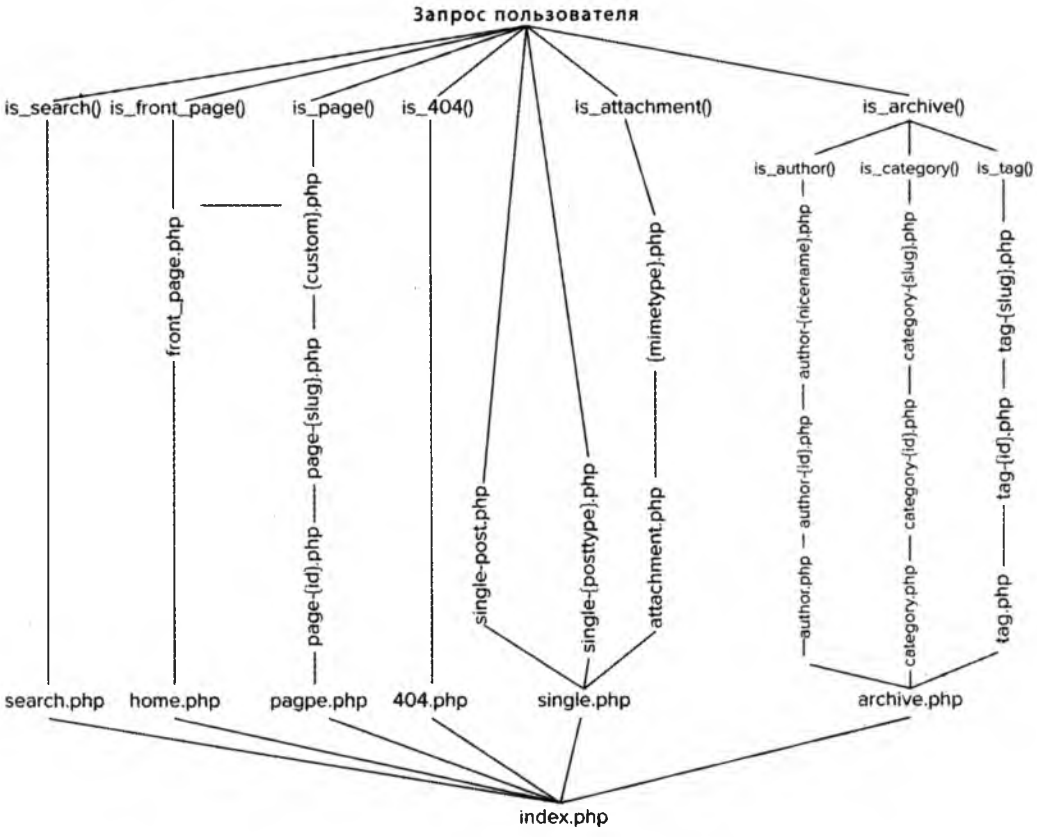


Рис. 9.3. Иерархия шаблонов WordPress

Создание собственной темы: дополнительные файлы

Трудно рассортировать различные файлы шаблонов, среди которых одни являются критически важными, другие существенными, а третьи просто неплохо иметь в наличии. В каждой теме набор шаблонов будет разным, подобранным, чтобы удовлетворять заданным автором целям для контента или дизайна. Действительно критичные и важные шаблоны мы уже обсудили. В некоторых кругах приведенные далее шаблоны также попадут в эти категории, так что тут вам нужно самим принимать решение. Не думайте, что из-за того, что речь о них идет позднее, эти шаблоны менее важны, чем остальные. Рассматривайте каждый тип шаблона как инструмент. Значение имеет то, как именно вы его используете.

Разберемся с ошибками 404: 404.php

Страница 404 — неизбежное зло. Рано или поздно посетители обнаружат что-нибудь устаревшее. В отличие от традиционных веб-сайтов, WordPress действительно

помогает избежать этого, поскольку обычно все элементы навигации динамически создаются существующим на данный момент контентом. Но вероятность того, что пользователь найдет нерабочую ссылку, остается, так что страница 404 неизбежно появится.

Тема Twenty Eleven предоставляет действительно хороший вариант шаблона 404 с включением поисковых полей, последних записей, наиболее популярных рубрик и облака меток. Таким образом, у посетителей, споткнувшихся об эту страницу, появляется возможность найти то, что они ищут.

Другой хороший вариант — отображение списка возможно связанного контента в виде: «Найти то, о чем вы просили, не вышло, но возможно, вас заинтересует что-либо из следующей информации». Вам не нужно, чтобы страница 404 оказалась тупиком. Всегда предлагайте просмотреть что-нибудь еще, обеспечивайте выход.

Наш магазин отправляет разработчику предупреждение о том, что кто-то запросил несуществующий URL, по электронной почте или через Twitter. По отсылкам к заголовкам HTTP вы можете отследить, откуда взялась битая ссылка. Как минимум, вы будете знать, что что-то пошло не так, и сможете поискать, в чем дело.

Также неплохо, если страница 404 забавная. Юмор — хорошее лекарство, которое стоит применять, чтобы обезоружить посетителей, расстроенных тем, что они не нашли то, что им нужно. Хороший вариант: показывать разработчикам ошибки, а посетителям — что-то полезное и осмысленное. Вспомните о днях кита-ошибки в Twitter. По мере роста Twitter у него частенько случались проблемы с расширяемостью и кит-ошибка появлялся куда чаще, чем сейчас. Но будучи таким веселым, кит-ошибка Twitter быстро стал иконкой в Интернете и обрел собственных почитателей.

Другая ошибка, которую обязательно нужно скрывать от посетителей — пусть и необязательно посредством шаблона, — это ошибка соединения с базой данных. В предустановленном варианте эта ошибка выглядит уродливо и дает слишком много информации посетителю, который, будем надеяться, хороший мальчик (или девочка) и не станет использовать ее во вред вашему веб-сайту.

В версии WordPress 2.5 появилась новая функция, позже транслированная и в предыдущие версии, где при ошибке соединения с базой данных WordPress ищет файл `db-error.php` в директории `wp-content`.

ЗАМЕЧАНИЕ

Этот файл находится вне директории темы. Поскольку соединения с базой данных нет, WordPress не знает, какую тему отображать.

Вы можете поместить в шаблон `db-error.php` любой код или CSS, за исключением динамичных данных функций WordPress, поскольку они не будут работать без базы данных. Это еще одна ситуация, из-за которой мы помещаем `db-error.php`

на все наши сайты на WordPress, обеспечивая симпатичное сообщение об ошибке и оповещая о ней команду разработчиков.

Вот пример файла db-error.php:

```
<?php
//error_reporting('E_ERROR');
mail('developers@mysite.com','WP SQL Connection Issue on '.$_SERVER['HTTP_HOST'],
'This is an automated message from the wordpress custom db error message file.');
?>
<html>
<head>
<title>Temporarily Unavailable</title>
<style>
body { background-color: #000; }
#wrapper
{
    width: 600px;
    height: 300px;
    margin: 2em auto 0;
    border: 4px solid #666;
    background-color: #fff;
    padding: 0 2em;
}
p { font-size: larger; }
</style>
</head>
<body>
<div id="wrapper">
    <center>
    <!-- /* This is the generic database error page that will be shown when a fatal
 db connection issue arises */ -->
    <h1><?php echo $_SERVER['HTTP_HOST']; ?> is Temporarily Unavailable</h1>
    <p>The webmaster has been alerted. Please try again later.</p>
  </center>
</div>
</body>
</html>
```

В редких случаях, когда WordPress не может подключиться к базе данных MySQL, пользователи увидят не уродливую ошибку базы данных, а симпатичное сообщение о ней, а разработчики получат электронное письмо. Это также позволяет информировать посетителей о том, что с их стороны никаких действий не требуется, кроме как вернуться позже, поскольку ошибка произошла на сервере веб-хостинга, а не у них. Опасность только в том, что если дела пойдут совсем плохо, разработчики потонут в письмах.

author.php

Раньше вы группировали контент по рубрике или метке. Но можно также использовать и шаблон author.php, чтобы отобразить единовременно весь контент, созданный одним автором. Кроме того, вы можете создать особый шаблон автора на основе его ID или никнейма, то есть имени пользователя. Как всегда, пригодный

для чтения человеком шаблон на основе имени предпочитается в WordPress основанному на ID.

Иногда на сайте есть несколько авторов, как в случае с сайтом команды разработчиков. В подобных случаях посетитель может захотеть найти другие записи, сделанные тем же человеком. Шаблон `author.php` отображает только записи, созданные определенным автором.

Шаблон автора ведет себя как цикл рубрики и метки. Приятной особенностью темы Twenty Eleven является то, что он также включает любую информацию об авторе, которую тот внес в своем профайле.

```
<?php
// Если пользователь заполнил поле биографии,
// то биография будет отображаться с записями пользователя.
if ( get_the_author_meta( 'description' ) ) : ?>
  <div id="author-info">
    <div id="author-avatar">
      <?php echo get_avatar( get_the_author_meta( 'user_email' ),
        apply_filters( 'twentyeleven_author_bio_avatar_size', 60 ) ); ?>
    </div><!-- #author-avatar -->
    <div id="author-description">
      <h2>
        <?php printf( __( 'About %s', 'twentyeleven' ), get_the_author() ); ?>
      </h2>
      <?php the_author_meta( 'description' ); ?>
    </div><!-- #author-description   -->
  </div><!-- #author-info -->
<?php endif; ?>
```

Если авторы дают какие-то биографические данные, эта информация публикуется здесь. Функциональность можно расширить, если на странице профиля есть редактор обогащенного текста для биографической информации и, возможно, какие-то расширенные произвольные поля.

На производственных сайтах такие поля могут использоваться для создания сайта «Мультибизнес партнера», где каждый автор являлся, по сути, компанией. Также с использованием этого метода можно создать сайт типа Rolodex.

comments.php

Шаблон комментариев — один из самых сложных. Он имеет дело и с циклом комментариев, включая трекбэки и пинги, и с формой ввода для введения комментария посетителем, как авторизованным, так и анонимным. Хотя функционально эти задачи связаны, при сортировке в этом файле шаблона вы увидите много “if . . . else”, что сложно для темы. Тема может вовсе не включать в себя комментарии, особенно если вы используете WordPress как систему управления контентом, но если они есть, вы можете включить шаблоны функциональности комментариев в собственные шаблоны с помощью следующего кода:

```
<?php comments_template(); ?>
```

Вариантов комментариев существует бесконечное количество, на любой вкус и цвет, — возможностей чересчур много, чтобы обсуждать какую-либо из них отдельно. Но нужно не забыть при работе над этим шаблоном, что уже в WordPress 2.7 появилась возможность веток комментариев. Более подробная информация по использованию `wp_list_comments()` содержится в кодексе WordPress.

Также следует отметить, что в WordPress 2.7 цикл комментариев был упрощен, чтобы выглядеть в коде более схожим с традиционным циклом записи. Кроме того, появилось множество новых функций, чтобы сделать шаблоны комментариев более очевидными, и тема Twenty Eleven отлично их использует. Шаблон комментариев Twenty Eleven — прекрасная отправная точка, если вы хотите основательно пересмотреть раздел комментариев на сайте.

Одним особенным улучшением является функция `comment_form()`. Сейчас она отвечает за генерирование актуальной формы комментария в шаблоне. До нее все формы и логика форм находились непосредственно в шаблоне, что, естественно, добавляло сложности. Функция появилась в WordPress 3.0 и прекрасно расширяется, если вам нужно индивидуализировать форму комментария.

Как уже было замечено, шаблон комментариев может выбить почву из-под ног и является отдельной темой, потому что с ним много чего происходит. На некоторых сайтах просто продаются темы комментариев — например, на http://commentbits.com/. Это не полные темы сайтов, а отдельные шаблоны для комментариев с некоторыми вариациями. Это может быть простым способом стилизации подсистемы комментариев с быстрым запуском.

Добавление функциональности в шаблоны: functions.php

Шаблон `functions.php` не является отображаемым шаблоном, так что он не похож на остальные шаблоны, о которых мы говорили, но является крайне важным файлом, хотя и не отображает контент на веб-сайте впрямую. Преимущественно именно в файле `functions.php` готовятся «дрожжи», пробуждающие вашу тему к жизни. Это место, куда вы можете поместить то, что традиционно называлось «библиотечный код». Если в шаблонах вы обнаруживаете повторяющийся код или вам нужна особая функциональность, файл `functions.php` является тем местом, куда ее следует поместить. WordPress автоматически включает этот файл при выполнении, так что функции доступны во всех файлах шаблонов.

Нередко при добавлении функциональности вам нужно решить, относится код к `functions.php` или к плагину. Общее правило большого пальца: если вы добавляете что-то конфигурируемое и пользователь может захотеть отключить это без ущерба для внешнего вида сайта, это плагин. Если вы добавляете что-то существенное для темы, что всегда должно работать, это нужно помещать в `functions.php`.

`functions.php` очень хорошо комментирован. Для каждой функции и логического блока функции есть однострочный комментарий, объясняющий, что они делают.

Таким образом, `functions.php` легко менять и расширять. Однако большинство этих функций не стоит менять на работающем сайте, если только у вас нет специфической необходимости. Большинство из них просто добавляются к файлам шаблонов, чтобы создать богатый ловушками шаблон HTML для стилизации посредством CSS.

При добавлении к теме файла `functions.php` важно помнить, встраивается добавляемая функциональность в тему или же является независимой и относится к плагину. Это говорит о том, будет ли добавляемый код непосредственно применяться к теме или же он будет портативным и сможет применяться вне зависимости от того, какая тема используется на сайте. Это нередко может быть сложным выбором, особенно когда вы понимаете, что используете те же зацепки и фильтры, которые мы обсуждали в главе 8. По сути, файл функций является библиотекой плагинов, собранных в один файл.

Одна из основных задач файла `functions.php` — активировать или деактивировать определенные возможности WordPress в вашей теме. В теме Twenty Eleven все это делается с помощью функции `twentyeleven_setup()`. Вы увидите, что она активирует поддержку различных возможностей, включая форматы записей, произвольный фон, иконки для записей и некоторые другие. Все это — возможности WordPress, используемые темой Twenty Eleven; так что они должны быть активированы. Думайте об этом как о флажках возможностей. Как каждая возможность будет активирована и сконфигурирована, зависит от нее самой. Например, активировать иконки записей или изображения просто:

```
// Эта тема использует миниатюры, связанные с записью/страницей
add_theme_support( 'post-thumbnails' );
```

Некоторые флажки особенностей могут взять или запросить конфигурационную информацию:

```
// Добавляем поддержку различных форматов записи
add_theme_support( 'post-formats', array( 'aside', 'link', 'gallery', 'status',
  'quote', 'image' ) );
```

Однако каждая возможность отличается от остальных и должна быть активирована и сконфигурирована в зависимости от потребностей вашей темы.

Кроме того, файл функций устанавливает и определяет навигационные меню. В этом файле вы заявляете, сколько меню будет у темы, и приписываете им имена. Далее в этой главе мы поговорим о том, как определять местоположение меню в файлах шаблонов и задавать меню для этих локаций в консоли WordPress.

Так же как и меню, в файле `functions.php` вы можете определять и создавать зоны виджетов. В целом области виджетов требуют большей конфигурации, чем меню. Как и в случае с меню, вы регистрируете зоны виджетов или боковые панели в файле функций. Далее вы узнаете, как помещать эти локации в файлы шаблонов.

Файл `functions.php` предназначен для создания собственного рисунка поведения и функциональности темы. Вы можете использовать новую логику отображения

или новые особенности, подходящие для ваших потребностей и целей. Но вы также можете и переписать или изменить существующие возможности WordPress — например, в теме Twenty Eleven:

```
function twentyeleven_body_classes( $classes ) {
  if ( function_exists( 'is_multi_author' ) && ! is_multi_author() )
    $classes[] = 'single-author';
  if ( is_singular() && ! is_home() && ! is_page_template( 'showcase.php' )
       && ! is_page_template( 'sidebar-page.php' ) )
    $classes[] = 'singular';
  return $classes;
}
add_filter( 'body_class', 'twentyeleven_body_classes' );
```

Эти функции добавляются к функции `body_class()`, приписывающей классы CSS к узлу тела HTML. Предустановленная функция `body_class()` возвращает множество полезных классов, которые вы можете снабдить стилями CSS с помощью ловушек, но иногда вам требуется нечто особенное. В случае с темой Twenty Eleven предыдущая функция прицепляет классы к массиву, который возвращается на основе определенных критериев.

При изменении функций стартовой темы вам нужно делать выбор. Иногда, если вы вносите в файл минимальные изменения, вы можете просто подправить его напрямую, принимая тот факт, что это нарушит возможность темы обновляться. В противном случае вы можете добавить собственный `custom_functions.php` из файла `functions.php` и вносить все изменения здесь. Не забудьте, что, переписывая файл `functions.php` в случае обновления темы или ошибки пользователя, нужно поместить `include` обратно в этот файл, до того как вам удастся разбить голову, стучась лбом об стену по причине того, что произвольные функции не работают. На практике оба сценария работают успешно.

Мощь и уровень контроля, обеспечиваемые файлом функций, побуждают пойти вразнос, что может быстро привести к выходу размеров сайта из-под контроля. Например, в некоторых продвинутых темах и каркасах тем уровня «премиум» есть собственные консоли для изменения настроек темы. Такие каркасы рассматриваются в этой главе далее. Код консоли включает файл `functions.php`.

Например, рассмотрим популярный каркас темы Thematic. Для обеспечения управляемости и определенности файл `functions.php` в Thematic представляет собой просто список включений других файлов функций. Это логически разделяет и отделяет различные грани каркаса и оберегает файл от бесконечного разрастания. В эту тему также входит базовая консоль для контроля настроек темы.

Чтобы создать собственную консоль, нужно зарегистрировать функции контроля темы в WordPress. Кроме того, нужно создать форму HTML для консоли в файле `functions.php`. Это одна из причин того, что нам нравится способ, которым файлы разделены на отдельные зоны в Thematic. Смесь кодов PHP и HTML никогда не выглядит красиво или читаемо. Лучше держать информацию раздельно и в файлах разумных размеров.

Создание собственной консоли темы лежит за рамками этой книги, но это отличная возможность, и большинство каркасов тем класса «премиум» включает в себя эту функциональность. Наличие консоли темы помогает вашей теме WordPress перекинуть мостик от движка блога к полноценной платформе управления контентом для среднего пользователя. Однако новый настройщик тем, выпущенный в WordPress 3.4 и рассматриваемый в этой главе, может работать для многих сайтов как простая консоль темы. Для кодера WordPress легко расширяется посредством кода и широкой функциональности WordPress, но для среднестатистического пользователя, который, возможно, и является вашим клиентом, избегать кода критически важно, что делает эти контрольные панели и настройщики тем идеальными для конечного пользователя.

search.php

Файл шаблона поиска — ошибочное название. Этот шаблон, по сути, является страницей результатов работы поискового движка (search engine result page, SERP). Сама форма поиска называется `searchform.php` и рассматривается в следующем разделе. Идея страницы результатов работы поискового движка объясняет сама себя. Здесь должны отображаться результаты пользовательского поиска, по умолчанию — в обратном хронологическом порядке. В главе 11 рассматриваются некоторые слабые места встроенной поисковой функциональности WordPress и называются кое-какие альтернативы для улучшения пользовательского опыта.

Все это делает базовый поисковый шаблон Twenty Eleven. При наличии результата этот шаблон представляет их в браузере, при отсутствии — показывает новую поисковую форму.

Для улучшения предустановленного варианта страницы результатов работы поискового движка можно сделать пару вещей. Для начала не стоит делать тупик из этой страницы, если никаких результатов поиск не дал. Плагины могут предложить схожие запросы, в том числе и по созвучию, на основе изначального ввода. Все это сделает поиск более похожим на традиционную поисковую систему.

Если поиск по-прежнему не дает результатов, предложите альтернативный контент, который может заинтересовать пользователя, по аналогии с поведением предустановленного шаблона 404 в Twenty Eleven. Это может быть прекрасным местом для облака меток или списка наиболее популярного содержимого. Для демонстрации самого популярного контента есть плагины, или же вы можете создать произвольный запрос, только нужно определить его параметры.

Для некоторых сайтов проблема топ-контента уже решена, то есть мы выбрали, что будет таким содержимым. Для таких случаев была создана специальная рубрика записей и новый цикл для показа только этой категории на странице поиска.

При наличии результатов некоторые люди предпочитают видеть поисковые элементы выделенными на странице результатов работы поискового движка. В теме Twenty Eleven используется `the_excerpt()` для отображения выдержек контента

в результатах. Здесь и нужно внести изменения для выделения поисковых элементов. Обратная сторона разделения темы на много файлов шаблонов — необходимость поиска функций `include()` и `get_template_part()` для определения файла, подлежащего редактуре. В случае с Twenty Eleven `get_template_part()` ищет шаблон контента для правильного типа файла. Для ускорения процесса вы можете обратиться к `content.php` за общим контентом записи. В шаблоне `content.php` есть оператор `if` (он находится где-то в районе 35-й строки), проверяющий, будет ли контент отображаться на странице результатов работы поискового движка. Именно здесь тема решает, отображать весь контент или выдержку, так что вы можете изменить способ отображения выдержек.

Замените строку

```
<?php the_excerpt(); ?>
```

на:

```
<?php
  $excerpt = get_the_excerpt();
  $keys = explode(" ",$s);
  $excerpt = preg_replace('/('.implode('|', $keys) .')/iu',
    '<span class="searchTerm">\0 </span>',$excerpt);
  echo $excerpt;
?>
```

Поскольку `the_excerpt()` передает контент непосредственно на генерирование, можно использовать функцию API плагина `get_the_excerpt()`, возвращающую вместо этого строку. Прогоните эту строку через регулярное выражение `replace`, чтобы поместить элементы `span` вокруг всех поисковых элементов, а затем передать все это на генерирование. В CSS можно добавить приятное правило для выделения этих элементов `span` в соответствии с темой.

Наконец, если пользователи не нашли того, что искали, после просмотра результатов поиска, вместо того чтобы заставлять их прокручивать страницу назад, можно создать вторую поисковую форму внизу для уточнения поиска. После цикла результатов добавьте что-нибудь в духе:

```
<h2>Not seeing what you're looking for?  Try again</h2>
<?php get_search_form(); ?>
```

В теме Twenty Eleven поисковая форма уже активирована для вас. В главе 12 обсуждается улучшение способа работы поиска посредством плагинов и других альтернатив.

searchform.php

Обычная поисковая форма выдергивается из файлов шаблона в ядре WordPress и выглядит довольно просто. Если для темы нужно индивидуальное поле ввода поискового запроса, создайте новый шаблон с названием `searchform.php`. Эта форма может быть подогнана по стилю для соответствия остальной теме. Поисковый

виджет автоматически использует этот шаблон, чтобы включить форму в обычные шаблоны с помощью следующего кода:

```
<?php get_search_form(); ?>
```

Базовая форма поиска Twenty Eleven выглядит так:

```
<form method="get" id="searchform"
  action="<?php echo esc_url( home_url( '/' ) ); ?>">
  <label for="s" class="assistive-text">
    <?php _e( 'Search', 'twentyeleven' ); ?>
  </label>
  <input type="text" class="field" name="s" id="s"
    placeholder="<?php esc_attr_e( 'Search', 'twentyeleven' ); ?>" />
  <input type="submit" class="submit" name="submit"
    id="searchsubmit" value="<?php esc_attr_e( 'Search', 'twentyeleven' ); ?>" />
</form>
```

Обратите внимание, что поскольку эта форма может использоваться в ненумерованных списках боковых панелей, а также в любых местах, куда вы ее включите, пометки HTML нужно настроить так, чтобы они были универсальными.

Другая возможность для специальных поисковых форм, которую нередко можно увидеть в шапках сайтов, — это создание традиционного PHP `include` для поисковой формы. Убедитесь, что имя файла не находится среди имен, зарезервированных для движка шаблонов, а затем включите его в нужном месте в другой шаблон:

```
<?php include($bloginfo['template_directory'].'includeThis.php'); ?>
```

Не забывайте использовать массив `bloginfo[]`, чтобы сохранять тему мобильной. Вы также можете использовать этот метод для соответствия принципу DRY («не повторяйтесь»), когда однородные элементы расположены на разных страницах, но вне шаблонов заголовка и «подвала». WordPress сам делает большую работу, внедряя DRY в различные шаблоны страниц и заставляя вас как разработчика следовать правилам. Но на этом варианте свет клином не сошелся, так что тут вам могут помочь и традиционные операции с PHP. Эта функциональность часто используется для поддержания разумного размера шаблонов.

Другие файлы

Вот еще некоторые файлы, позволяющие довести тему до блеска. В консоль управления темами неплохо включить скриншот, чтобы тему легко было опознать по виду. Создайте изображение, представляющее тему, размером 300 × 225 пикселов и сохраните его как PNG. GIF и JPG также допустимы, иерархия предпочтений соответствующая. Обычно это изображение — скриншот сайта, использующего тему. Оставшаяся информация по каждой теме на странице управления темами берется из заголовка `style.css`.

Многие темы включают в себя несколько языковых файлов и сразу готовы к локализации. Если вы собираетесь запускать сайт на нескольких языках, обратите

внимание на ограничения. Локализация и интернационализация лежат за рамками нашей книги. Просто помните, что WordPress поддерживает эту функциональность.

Шаблоны произвольных страниц

Время от времени у вас будут возникать особые страницы, требующие уникальной разметки, отличной от остального сайта. Это может быть страница с контактной информацией или веб-сайт в виде брошюры, где у каждого товара своя уникальная страница. Возможно, вы делаете отдельную страницу для маркетинговой кампании или QR-кода. Возможно, использование общего шаблона `page.php` не соответствует потребностям сайта, поскольку каждая страница наделена неповторимыми свойствами. Может, какие-то виджеты вы хотите отображать на определенных страницах, но не на всех, хотя этот вопрос можно решить с помощью плагина, такого как Widget Logic. Или же вы интегрируете веб-приложение стороннего автора с сайтом WordPress. Тут вступают в дело шаблоны страниц.

Вы можете приписать шаблоны страниц странице, используя вкладку **Создать** в консоли администратора. WordPress задаст шаблон для страницы при отображении контента в соответствии с уже установленной схемой специфичности. Например, если странице приписывается шаблон страницы, он выбирается, поскольку представляет собой наиболее подходящий вариант. Если для страницы задан предустановленный шаблон, для генерирования контента будет использован шаблон `page.php`. Наконец, если ни один из этих шаблонов не доступен, WordPress использует шаблон `index.php`.

На рис. 9.4 представлены несколько шаблонов страниц на выбор, включая предустановленный, шаблоны Twenty Eleven для страницы с демонстрацией и боковой панели, о которых речь пойдет далее, и два шаблона, добавленных для примера: Веселый и Скучный.

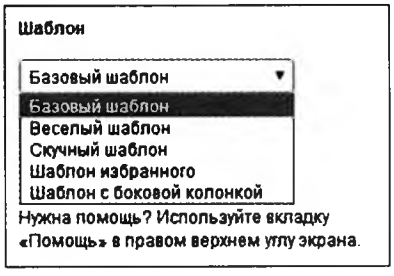


Рис. 9.4. Выбор шаблона страниц

Когда использовать шаблоны произвольных страницы

Для использования шаблонов произвольных страниц на сайте есть множество причин. Это весьма мощный инструмент из вашего арсенала, и при грамотном применении с его помощью можно значительно расширить сайт. Произвольные шаблоны страниц — еще один способ приписать шаблоны тем или иным страницам. В отличие от предыдущих примеров, основанных на атрибутировании страниц по слагу или ID, рубрике или метке, произвольные шаблоны могут быть заданы для любой страницы на сайте через вкладку **Создать**.

Простым примером является создание шаблонов для уникальных страниц товаров, на боковой панели каждой из которых отображаются уникальные данные

и ссылки. Как и всегда с WordPress, есть много способов реализовать эту функциональность, но иногда все, что нужно при создании произвольных шаблонов страниц, — это простой и прямой метод. Часто простое оказывается лучшим.

Другой простой пример — создание шаблона произвольной страницы, использующего элемент HTML iframe для включения веб-приложения стороннего разработчика. В зависимости от потребностей и задач сайта (не говоря о бюджете) это может быть простым и грубым способом интеграции двух сайтов в один. Побочные эффекты этого метода такие же, как и всегда при использовании iframes, то есть сомнительный внешний вид. Но общеизвестно, что метод этот использовался и ранее, потому что иногда быстро и грубо — это все, что нужно.

Среди более сложных примеров — интеграция различных веб-приложений с сайтом на WordPress. Например, шаблон страницы может использоваться для создания страницы произвольного заказа, который заносится непосредственно в пакет электронной коммерции. Установка и поддержка такой штуки через консоль WordPress могли бы стать сущим кошмаром, но при использовании произвольных шаблонов это просто код, при этом все блага WordPress остаются при вас.

В реальной жизни шаблоны произвольных страниц используются для внесения в календарь событий и регистрации. Как-то раз был создан дорогой учебный проект, предлагающий веб-приложения для поиска и отображения курсов, а также регистрацию для посещения лично или через веб. Самым простым способом интеграции этой системы регистрации для обучения с клиентскими сайтами на WordPress оказалось создание шаблонов произвольных страниц.

По сути, существующая система регистрации была расширена некоторыми командами веб-сервиса REST. Затем был создан набор шаблонов произвольных страниц, вступающий в контакт с веб-сервисами, но отображающий контент локально внутри сайта WordPress с применением локального листа стилей. Хотя установка шаблонов страниц поначалу устрашала, ее преимущества оказались огромными:

- Продолжение использования существующей системы, с которой уже знаком персонал.
- Расширенные возможности регистрации на разных сайтах, увеличивающие потенциальную аудиторию.
- Несмотря на распределение регистрационных форм по различным сайтам, они остались частью централизованной системы.
- Образовательная система соответствует виду локального сайта, поскольку она использует локальную тему WordPress.

Как использовать шаблоны произвольных страниц

Создание шаблонов произвольных страниц не представляет никакой сложности. Дело усложняет цель их создания и методы ее достижения.

Чтобы создать шаблон страницы, скопируйте существующий шаблон, схожий с тем, который вы хотите сделать. Обычно это шаблон `page.php`. Назовите новый файл шаблона как хотите и поместите его в директорию `theme`. В наших магазинах разработок мы следуем правилу, называя шаблоны страницы `t_templatename.php`. То есть названия содержат префикс `t_`, так что их легко отличить от традиционных файлов шаблонов, хотя имя этого файла не имеет значения, пока вы избегаете имен, зарезервированных в иерархии.

Чтобы сделать новый шаблон шаблоном страницы, нужно добавить особый комментарий в верхнюю часть файла:

```
<?php
/*
Template Name: Fancy Page Template
*/
?>
```

Он должен быть в первой паре строк файла, чтобы WordPress просканировал и зарегистрировал файл как шаблон страницы. На практике выше этого изречения имеется только комментарий по управлению исходным кодом.

Имя шаблона может быть любым. Оно должно иметь смысл, но быть не слишком длинным, поскольку WordPress использует этот комментарий PHP в выпадающем меню в консоли. Теперь шаблон страницы зарегистрирован в WordPress.

Оставшаяся часть шаблона страницы может содержать все, что необходимо для достижения задач, поставленных перед вашей темой. Вы можете — и скорее всего должны — использовать встроенные функции WordPress, такие как `get_header()` и `get_footer()`, а также сборщики контента. По сути, тут вы делаете все, что хотите, не забудьте только, что эту яму вы роете для себя.

Например, если вы уберете динамические боковые панели WordPress и замените их статическим HTML, вы также удалите из консоли функциональность, позволяющую управлять виджетами в этом шаблоне. Лучше зарегистрировать новую область виджетов в этом шаблоне и продолжить использовать консоль для управления контентом.

Помните, что шаблонам страниц не запрещается отображать информацию страницы. Вы можете создать шаблон, отображающий традиционный цикл, или сделать что-то, совершенно не связанное с контентом WordPress. Тогда просто оставьте поле в редакторе страницы пустым или внесите туда памятку для себя любимого.

Шаблоны страниц в Twenty Eleven

Тема Twenty Eleven стандартно поставляется с двумя шаблонами страницы для использования на сайте. Раз уж они есть, рассмотрим их вкратце.

Первый шаблон страницы, поступающий с темой Twenty Eleven, — это `sidebar-page.php`. Он довольно простой и понятный. По сути, он добавляет в шаблон страницы боковую панель, что ясно из названия.

Второй произвольный шаблон страницы сложнее. Он называется `showcase.php`. Шаблон демонстрационной страницы спроектирован, чтобы быть интересной титульной страницей сайта на WordPress. В него встроена пара возможностей.

Во-первых, у него есть выделенная зона контента. Она появляется непосредственно под навигацией Twenty Eleven. Контент поступает из содержимого страницы, которой вы приписали этот шаблон.

Во-вторых, у него есть демонстрационное поле для записей. Это примерно то же самое, что было сделано ранее в этой главе для слайдов и слайд-шоу, но в случае с Twenty Eleven используется контент записи. Чтобы использовать записи таким образом, нужно сделать их приклеенными. Приятная мелочь: эта возможность поддерживает смешанные форматы, то есть в одной области могут быть изображения, тексты или одновременно изображения и тексты.

Как только вы опубликовали эту страницу, установите ее как титульную в настройках чтения в консоли, и у вас будет замечательная новая стартовая страница.

Шаблоны произвольных страниц — мощный инструмент. В самом деле, если вы не можете вместить контент в предустановленные типы шаблонов, у вас всегда есть этот козырь в рукаве, чтобы создать шаблон произвольной страницы и перезаписать все, что угодно. Это также отличный способ добавления на сайт функциональности, не относящейся к WordPress.

Другие расширения темы

Вот некоторые дополнительные расширения, которые можно применить к теме. Большинство из них приводят в итоге в файл функций и не являются индивидуальными файлами. Они позволяют обогатить возможности администратора сайта для управления контентом.

Управление меню

Как мы уже отмечали, сайты WordPress, использующие систему управления контентом (CMS), представляют собой комбинацию страниц и записей. Самая сложная часть такой гибридной разметки — создание осмысленной навигации. Общая навигация по сайту — важный момент, если вы рассчитываете на постоянство посетителей. Посетители должны иметь возможность исследовать сайт органично, на собственном опыте, пользуясь связанными записями и страницами, но стратегия организации контента также должна быть налицо, — это и есть задача общей навигации. Иногда нам везло с сайтом, структура которого позволяла создать два уровня навигации: для контента страниц и для контента записей. До выхода WordPress 3.0 два типа контента требовалось сливать вместе, а навигацию кодировать вручную с помощью функций `wp_list_pages()` с множеством параметров. Все это кодировалось в шаблонах и было совсем негибким с точки зрения администратора сайта.

Однако то время прошло. В WordPress 3.0 появилась новая система управления меню. Она передала весь контроль администратору сайта. Как разработчику темы вам нужно просто ее установить.

Сначала в шаблоне `functions.php` нужно активировать функцию меню. Это делается так:

```
if (function_exists('add_theme_support')) {
    add_theme_support('menus');
}
```

Затем нужно зарегистрировать меню для темы. По сути, это означает приписывание названных локаций для каждого меню, чтобы пометить «собственность» HTML для меню. Используется функция `register_nav_menu()`. Она принимает два параметра. Первый — это никнейм, или идентификатор, используемый в файлах шаблонов. Второй — читаемое имя, используемое в консоли WordPress. Например, вот так создается единственное общее навигационное меню для использования в теме:

```
register_nav_menu('primary', 'Global navigation menu');
```

В теме может быть столько мест расположения меню, сколько вам хочется. Просто скармливайте функции `register_nav_menu()` массив локаций:

```
register_nav_menus( array(
'primary' => __( 'Primary Navigation', 'twentyten' ),
'supernav' => __( 'Super Navigation', 'twentyten' ),
) );
```

Таким образом WordPress поймет, что локации меню доступны и идентифицированы, но вам также нужно прописать их позицию в файлах шаблонов. Обычно это происходит в шаблоне `header.php`. Чтобы поместить желаемое меню в файл шаблона, используйте функцию `wp_nav_menu()`:

```
<?php wp_nav_menu( array( 'theme_location' => 'primary' ) ); ?>
```

Она может принимать много параметров, переданных как массив, чтобы контролировать стиль HTML, но важный параметр — определить, какое меню вы хотите поместить в этой локации. В предыдущем примере участок зарезервирован по меню с названием `primary`.

Наконец, чтобы собрать все воедино, администратор может использовать консоль меню. Обратите внимание, что вы определяете местоположение меню, а затем задаете этой локации актуальную позицию в иерархии HTML. Все это делается на уровне кода. В свою очередь, администратор сайта задает контент для этих локаций через консоль. То есть, несмотря на то что все это устанавливается вместе и работает вместе, компоненты в итоге разъединены. Определение осмысленных имен для локаций меню важно в отношении управления меню через консоль, но администратор может использовать меню совсем не так, как вам думается.

Что значит «разъединены»? Давайте вернемся к тем временам, когда WordPress создавал меню без новой системы. Процесс требовал использования функции

`wp_list_pages()` с множеством параметров для создания вручную необходимого меню. В первом издании нашей книги много страниц было посвящено именно этой теме. Программное создание меню в файле шаблона работало потому, что оно было непосредственно связано с актуальным контентом сайта и иерархией страниц.

Однако вариант не был идеальным. Необходимо было прибегать к хитростям, таким как создание пустой страницы для появления в автоматизированном меню навигации и последующее использование плагина Page Links To, написанного Марком Джекитом, для перенаправки страницы в шаблон рубрики или архива. Кроме того, для управления иерархией и порядком страниц для функции `wp_list_pages()` рекомендовалось использовать плагин PageMash. Этот проверенный метод работает для множества сайтов. Работает он потому, что является программным, то есть предсказуемым, потому что привязан непосредственно к контенту. Иногда — не работает, если контент меняется слишком сильно.

Иногда изменения в контенте требуют отладки параметров `wp_list_pages()`. При использовании данного метода изменения нужно вносить на уровне кода, что ограничивает возможности администратора сайта. Хорошо это или плохо — вопрос, на который вы отвечаете исходя из условий и потребностей сайта.

Предсказуемость новой системы управления меню — еще одна проблема. С ней навигация сайта является произвольной. Она не привязана к контенту. Вы можете автоматически добавлять страницы в меню верхнего уровня, но не можете автоматически добавлять дочерние страницы. Кроме того, в системе меню не отображается иерархия страниц. Меню приходится управлять вручную, помимо управления контентом. Одна из возможных ловушек — оставшийся после удаления страницы пункт меню.

Есть и «за» и «против» такого разъединения. С одной стороны, у администратора сайта действительно появляется полный контроль над контентом меню. Он может подогнать меню под свои нужды. Однако важно, чтобы администратор сайта понимал, что он должен управлять и контентом, и меню. Могут возникнуть сложности, если некоторые аспекты навигации созданы программно в шаблонах темы, а меню при этом состряпано вручную.

Например, представьте, что вы создали для клиента новую тему с множеством страниц товаров. Это особые страницы, подчеркивающие индивидуальность каждого товара, со специфической информацией и подробностями. Для дальнейшего улучшения пользовательского опыта и, возможно, перекрестной продажи каких-либо товаров вы должны создать шаблон страницы для товаров со специальным разделом дополняющих товаров внизу страницы. Этот раздел генерируется кодом в воображаемом шаблоне произвольной страницы `t_product_page.php`. Сложность в том, что когда администратор сайта добавляет страницу нового товара, она автоматически появляется в этом коде страницы подходящих товаров на некоторых других страницах, но не возникает автоматически в меню. Администратору сайта приходится вручную редактировать меню для ее добавления.

Это не катастрофа. Просто нужно разъединение и дополнительный шаг. Сложность в том, чтобы наделить администратора сайта соответствующими правами

и попросить его конфигурировать кое-что вручную или же просто добавить код, благодаря которому все автоматизируется.

Области виджетов

Области виджетов работают примерно так же, как меню. В файле `functions.php` вы определяете и называете различные области для различных частей сайта. Часто речь идет о боковых панелях, но возможностей гораздо больше. Не нужно ограничивать себя, думая о том, что виджеты могут находиться только на боковых панелях сайта. Во многих темах, включая Twenty Eleven, число и местоположение областей виджетов расширено и включает в себя «шапку», или область заголовка, «подвал» и даже середину цикла. Вы видели темы с сотнями областей виджетов.

Приятная особенность областей виджетов — их гибкость. Число виджетов, которые можно поместить в область виджетов, огромно. Наличие нескольких областей виджетов дает администратору сайта возможность контролировать эти области сайта, лежащие за рамками зоны первичного контента.

Как уже было замечено ранее, установка областей виджета схожа с установкой меню. В файле `functions.php` нужно определить и назвать локации. Поскольку в зонах виджетов контент более гибкий и разнообразный, чем в меню, в том числе в них много ненумерованных списков, вы обычно передаете некоторую информацию по HTML для использования при генерировании виджетов. Также, поскольку виджеты развились из традиционного использования на боковых панелях, функция их регистрации по-прежнему называется `register_sidebar()`, но пусть это вас не смущает. Вот фрагмент кода области виджетов из Twenty Eleven:

```
register_sidebar( array(
  'name' => __( 'Main Sidebar', 'twentyeleven' ),
  'id' => 'sidebar-1',
  'before_widget' => '<aside id="%1$s" class="widget %2$s">',
  'after_widget' => "</aside>",
  'before_title' => '<h3 class="widget-title">',
  'after_title' => '</h3>',
) );
```

Как и в случае с меню, вы передаете функции удобоваримое имя для использования в консолях и идентификатор для использования в шаблонах темы. Помимо этого есть дополнительные параметры для единообразной стилизации виджета. В этом примере Twenty Eleven использует элементы HTML5 `aside`, чтобы заключить каждый виджет и его название в теги `h3`. Этот код уведомляет WordPress о том, что для контента доступна область виджета.

Следующий шаг — задание HTML позиции пространства для этой области виджета. В данном примере мы смотрим на `sidebar.php` в Twenty Eleven, но, повторим, области виджета могут быть в любом месте шаблона. Вот код:

```
<?php if ( ! dynamic_sidebar( 'sidebar-1' ) ) : ?>
  <aside id="archives" class="widget">
```

```
    <h3 class="widget-title"><?php _e( 'Archives', 'twentyeleven' ); ?></h3>
    <ul>
      <?php wp_get_archives( array( 'type' => 'monthly' ) ); ?>
    </ul>
  </aside>
  <aside id="meta" class="widget">
    <h3 class="widget-title"><?php _e( 'Meta', 'twentyeleven' ); ?></h3>
    <ul>
      <?php wp_register(); ?>
      <li><?php wp_loginout(); ?></li>
      <?php wp_meta(); ?>
    </ul>
  </aside>
<?php endif; // конец боковой панели с виджетами?>
```

В этом фрагменте кода происходит пара вещей. Главная: в первой строке определяется, назначен ли в консоли контент для области виджета с названием `sidebar-1`. Если да, он будет отображен. Это одно из крупных преимуществ областей виджетов: при отсутствии контента они не отображаются, если только вы не следуете парадигме, показанной в предыдущем коде.

В таком случае, если для области виджетов виджетов нет, в ней отображается предустановленный контент. В предыдущем примере в случае отсутствия динамического контента, заданного в консоли, в области виджета автоматически отображаются виджет архивов и метавиджет. Как именно вы все настроите, зависит от ваших потребностей. Вы можете просто обеспечить области виджетов без предустановленного содержимого, чтобы расширить возможности темы.

Форматы записей

Это простая функция с флажком включения/выключения в файле `functions.php`. Если в теме предполагается использование встроенных форматов записей или дополнительного набора, ее можно активировать.

Форматы записей — это преимущественно способ индивидуализации отображения определенных типов записей. Обычно это применяется при использовании WordPress для ведения блога или архива журнала. Различные форматы записей могут использоваться для получения различного генерирования HTML в циклах, как мы показали ранее в этой главе. Это позволяет показывать цитаты или ссылки не так, как полные записи.

На данный момент WordPress поддерживает десять форматов записей для разного контента. Вы можете активировать все или некоторые из них, в зависимости от целей, для которых создается тема. Стандартный формат предназначен для традиционных записей и активируется автоматически. Он использовался в WordPress всегда, а имя получил только сейчас.

Кроме того, начиная с WordPress 3.1 есть девять новых форматов. Вот их список с рекомендациями по стилю, хотя с учетом гибкости WordPress вы можете стилизовать их под свои нужды:

- **Aside.** Аналог быстрого добавления записи. Обычно идет без названия записи.
- **Audio.** Обычно этот формат используется для аудиофайла, например для подкаста или выпускаемого группой сингла.
- **Chat.** Этот формат обычно соответствует расшифровке чата. Стиль обычно задается с использованием HTML элемента `pre` для сохранения разрыва строк.
- **Gallery.** Формат для галереи приложений-изображений. В записи обычно содержится сокращенный код галереи.
- **Image.** Формат единичного изображения, которое может быть встроено в контент записи, или же URL внутри записи размещает сообщение на сайте.
- **Link.** Формат для ссылки на другой URL, отображающийся как простая ссылка без названия. Этот формат часто используется для создания сайта с закладками или напоминаний об интересных URL.
- **Quote.** Формат, предназначенный для цитат. Этот формат можно использовать без названия или же использовать название для атрибутирования цитаты.
- **Status.** Воспринимайте этом формат как обновление в Twitter или Facebook. Обычно используется без названия. Данный формат позволяет сделать собственный клон Twitter.
- **Video.** По аналогии с изображением или аудиозаписью демонстрирует посетителям отдельный видеоролик. Может быть встроен в контент или является внешним URL.

Чтобы активировать форматы записей в теме, просто выберите, какие форматы вы хотите использовать, и передайте их как массив параметров. Например, тема Twenty Eleven поддерживает набор названных выше параметров:

```
add_theme_support( 'post-formats', array( 'aside', 'link', 'gallery',
  'status', 'quote', 'image' ) );
```

Помните, что формат стандартной записи для традиционных записей всегда остается при вас. Форматы записей могут подогнать тему под специфическую нишу или расширить ее для демонстрации различных типов контента уникальными способами.

Настройки темы

Многие из каркасов тем предлагают специальную консоль настроек темы для индивидуализации каркаса. Это кодированное решение, исходящее от разработчика темы и создающее консоль с любыми настройками, которые он захотел включить, — еще одна возможность сбалансировать возможности администратора сайта, противоположная работе в коде шаблонов темы.

Создание собственной консоли настроек темы — довольное сложное предприятие, лежащее за рамками темы нашей книги. Каждая тема уникальна, и это дело разработчика, определяющего, какие особенности или аспекты могут настраиваться.

Чтобы создать консоль настроек темы, сначала нужно построить код консоли и формы и зарегистрировать их в WordPress. После этого нужно использовать конфигурируемые параметры в файлах шаблонов. Контроль, передаваемый администратору сайта, может быть весьма существенным, но тогда разработка шаблонов становится несколько более хитрой.

Больше информации по созданию консоли настроек темы содержится в кодексе WordPress и в каркасах других тем. В Интернете есть много хороших учебников.

Тонкий настройщик темы

Некоторая часть функциональности консоли темы была передана новому тонкому настройщику темы (Theme Customizer), появившемуся в WordPress 3.4.

Это новая консоль для тонкой настройки темы. Приятно, что она может сделать для администратора «живой» предварительный просмотр сайта в ходе настройки. Это в самом деле большое преимущество данного компонента. Просмотр в реальном времени позволяет администратору сайта экспериментировать и видеть результат без воздействия на работающий сайт до окончания изысканий. Раньше этот процесс включал внесение изменений в консоли настроек темы, о которой говорилось ранее, в коде или в других настраиваемых местах и их публикацию на работающем сайте. Затем администратор должен был просмотреть сайт, чтобы увидеть изменения. В то же время посетители, оказавшиеся на сайте, видели вносимые изменения непосредственно в работе. А изменения могли и не быть тем, чего ждал администратор, или, что еще хуже, испортить какие-то аспекты отображения. Предварительный просмотр в реальном времени обеспечивает весь этот процесс наряду с его контролем.

Тема Twenty Eleven — первая тема, в которой для любых существенных расширений используется этот новый набор функций, хотя и ожидается, что к моменту выхода этой книги он войдет во многие темы.

Изначально тонкий настройщик темы использует API настроек WordPress для хранения информации по конфигурируемому дизайну. Затем с помощью специально сделанных функций в файле `functions.php` WordPress применяет изменения к HTML.

В функции Twenty Eleven для изменения цвета заголовка файл `functions.php` добавляет новую строку CSS в заголовок HTML, которая переписывает лист стиля. На практике это не слишком оптимальный код для сайта с большим трафиком. Однако это еще один пример, когда оптимизацией жертвуют ради настроек, которые может делать администратор сайта.

Иерархия тем и дочерние темы

До сих пор в этой теме мы рассматривали файлы шаблонов, составляющие целую тему, на примере темы Twenty Eleven. Мы даже рассмотрели ее переименование

и создание произвольной темы в полученной новой директории с использованием Twenty Eleven в качестве стартовой темы. Это хороший способ начать создание темы, поскольку у вас есть возможность погрузиться в тонкости ее работы. И в реальном мире множество команд разработчиков работают именно так. Данный метод работает хорошо, поскольку вы точно знаете, где находятся файлы шаблонов и CSS, которые нужно редактировать. Тема — вещь в себе, что минимизирует рабочий процесс и усилия по отладке. Она, конечно, не совершенна, но весьма хороша.

Однако с выходом WordPress 2.7 функциональной реальностью стали дочерние темы. Хотя применять их можно было и ранее, до возможности наследования файла шаблона дочерняя тема не была жизнеспособным вариантом разработки. Дочерние темы позволяют взять существующую тему или каркас темы и использовать лучшее оттуда, после чего расширить и изменить каркас в соответствии с лицензией, чтобы достичь соответствия целям разрабатываемой темы, в то же время сохраняя возможность обновления через родительскую тему. Оставив в стороне основы разработки темы, вам лучше выбрать удобный для вас каркас (некоторые будут названы далее) и создать дочерние темы. Дочерние темы — будущее разработок для WordPress.

Концепция вполне революционна по нескольким причинам. Во-первых, она открывает двери для каркасов тем. Начиная с крепкого фундамента вы можете создать бесчисленное количество вариаций, просто наследуя теме. Каркасы темы, конечно, весьма плоские, и это не случайно. Но, используя дочерние темы, вы наследуете все семантические ловушки CSS и микропрелести и лепите вокруг них «конфетку», по сути, взяв самое лучшее и создав нечто новое.

Кроме того, обновление родительской темы или каркаса темы не перезапишет ваши настройки. Ранее при внесении изменений в копию темы необходимо было их отслеживать, чтобы вернуть на место после обновления темы. Это можно было до некоторой степени автоматизировать посредством решений для управления исходным кодом, но даже если все это работало, было непросто. К тому же рано или поздно забываешь внести изменения в обновленные файлы.

Наконец, дочерние темы проложили путь к автоматическому обновлению через менеджер тем. Постепенно шаблоны тем становятся уязвимыми с точки зрения безопасности, например для межсайтового скриптинга. Использование правильно созданных дочерних тем означает, что родительская тема может автоматически обновляться для устранения проблем с безопасностью, не затрагивая дочернюю тему. Это позволяет сделать сайт более безопасным.

Тут есть пара сложностей. Функциональность, сохраняющая настройки дочерней темы незатронутыми, работает в обе стороны. Если вы переписываете определенный файл шаблона собственными настройками, любые изменения в родительском файле шаблона с тем же именем не будут переданы вашему уникальному файлу. На практике это может создать ложное чувство безопасности, поскольку вы могли скопировать для изменений слабо кодированный файл, а изменения, внесенные в родительскую версию, оставят ваш файл нетронутым и по-прежнему уязвимым. Поэтому, сохраняя уязвимость в своем расширенном коде, вы продолжаете

перезаписывать им любой исправленный код. Это касается не только изменений по соображениям безопасности, но и любых полезных дополнений. Так что дочерняя тема не полностью избавляет вас от необходимости управлять кодом.

Кроме того, здесь имеет место некоторое главенство CSS. В целом дочерняя тема строится на CSS родительской темы. И в самом деле, именно так CSS и задуман работать, если мы вспомним о слове *каскадный* в его названии. Чтобы это работало в дочерних темах, они должны включать CSS родительской темы, и даже те правила, которые были в них переписаны. То есть размер CSS в дочерней теме может оказаться значительно больше, чем тот, что используется в браузере, но все это необходимо перенести.

В целом дочерние темы — фантастическая возможность WordPress, и мы советуем использовать эту методологию, если ситуация гарантирует ее работу. Конечно, поддержание чистого каркаса темы и его последующее расширение до индивидуальных сайтов добавляет преимущества языка общей темы CSS и функций, а также другие преимущества, названные ранее. Повторяем: использование дочерних тем — будущее разработки тем WordPress и лучшая из рекомендованных практик.

Давайте посмотрим, как работают дочерние темы и что нужно для создания такой темы. Сначала нужно найти тему, которая будет использоваться как родительская. Она не обязательно должна быть снабжена пометкой «каркас темы». Вы можете расширить любую тему, если она соответствует следующим требованиям:

- Лицензия позволяет расширять или изменять тему.
- Тема не является дочерней.

В данном примере мы продолжим строить на основе темы Twenty Eleven, но вы можете использовать любую тему или каркас в качестве родительской темы. Как говорилось ранее в этой главе, чтобы сделать произвольную тему дочерней или другой темой, нужно добавить строку в информацию в заголовке файла `style.css`. Эта строка информирует WordPress о местоположении родительской темы. Поэтому переменной в комментарии должно быть имя папки или темы. Хотя все зависит от сервера, лучше обращать внимание на регистр при наименовании темы. В данном случае мы добавляем следующую строку:

```
Template: twentyeleven
```

Полный блок комментария к заголовку в примере дочерней темы будет выглядеть следующим образом:

```
/*
Theme Name: A Twenty Eleven Child Theme
Theme URI: mirmillo.com
Description: A sample child theme
Author: David Damstra
Author URI: mirmillo.com
Template: twentyeleven
Version: 1.0
*/
```

Как уже обсуждалось ранее, наличие файла `style.css` с правильно сформатированной заголовочной информацией в папке с уникальным именем регистрирует тему в WordPress.

Следующий шаг — импорт CSS из родительской темы, чтобы у произвольной темы были базовые правила для работы:

```
/* импортируем базовые стили  */
@import url('../twentyeleven/style.css');
```

В этот момент можно активировать тему во вкладке **Внешний вид** в консоли WordPress. У вас есть полностью функциональная дочерняя тема на базе темы Twenty Eleven. Она выглядит точно так же, как тема Twenty Eleven, поскольку по сути это копия родительской темы. Остаток листа стиля работает как традиционный CSS, где вы можете переписать предыдущие правила правилами CSS в соответствии со специфичностью и первоочередностью, включая порядок, в котором они будут перечислены, — индивидуальные стили будут иметь преимущество, поскольку они появились позже.

Повторяем: работа CSS выходит за рамки этой книги, так что пойдем дальше и расширим немного дочернюю тему. Хотя все эти изменения можно сделать и с помощью тонкого настройщика, о котором говорилось ранее, в рамках данного примера мы внесем изменения в CSS. Сделаем для дочерней темы миленький розовый фон и изменим основной шрифт. Вот как может выглядеть файл `style.css` на выходе:

```
/*
Theme Name: A Twenty Eleven Child Theme
Theme URI: mirmillo.com
Description: A sample child theme
Author: David Damstra
Author URI: mirmillo.com
Template: twentyeleven
Version: 1.0
*/
/* импортируем базовые стили  */
@import url('../twentyeleven/style.css');
body {
    background: #E0A3BD;
    color: #333;
    font: 100%/1.5 calibri, arial, verdana, sans-serif;
}
```

С этого момента инструментарий разработчика браузера — ваш лучший друг. Используйте инспектор, чтобы увидеть текущие правила стиля, примененные к различным элементам, и внесите надлежащие изменения в CSS-файл дочерней темы. Повторяем: не забывайте следовать приоритетам и правилам специфичности CSS.

Дочерняя тема может быть настолько сложной или простой, насколько вы хотите. Вы можете создать совершенно уникальную тему, просто отредактировав лист стиля, как делали ранее. Или же дочерняя тема превращается в совершенно новую с новыми шаблонами, хотя в таком случае делать дочернюю тему, скорее всего, будет бессмысленно.

Вот как это работает. Когда WordPress решает, какой файл шаблона использовать, сначала он сканирует в его поисках директорию дочерней темы. Если там файла нет, сканируется директория родительской темы. WordPress предпочтет ваши файлы шаблонов файлам родительской темы, то есть вы можете переписать функциональность определенных шаблонов, оставив ядро родительской темы. Или же в дочерней теме могут быть шаблоны произвольных страниц, но основные шаблоны будут заимствованы из родительской темы. Тут есть великое множество возможностей, однако не забывайте о названных ранее ограничениях.

Самый простой способ сделать это — копировать файлы шаблонов, которые вы хотите изменить, из директории родительской темы в директорию дочерней темы, а затем изменить их как нужно.

Например, авторский шаблон в теме Twenty Eleven полностью функционален, но предположим, что вы хотите изменить размер изображения автора. Повторяем: это намеренно простой пример, и есть множество способов сделать это.

Сначала скопируйте шаблон `author.php` из темы Twenty Eleven в директорию дочерней темы. Затем отредактируйте дочернюю копию, изменив размер изображения примерно в 46-й строке кода. Получится примерно следующее:

```
<?php echo get_avatar( get_the_author_meta( 'user_email' ),
  apply_filters( 'twentyeleven_author_bio_avatar_size', 120 ) ); ?>
```

В данном примере вы удваиваете размер изображения с 60 до 120 пикселов по ширине. Пример того, на что все это будет похоже, можно увидеть на рис. 9.5.

Вы можете и дальше расширять дочернюю тему, используя файл `functions.php`. WordPress автоматически включает функции родительской темы, но, кроме того, он включает и функции темы дочерней. К наименованию функций нужно подходить с умом. Будьте осторожны, не создавайте в своей теме функции, название которых совпадает с функциями в родительской теме. Если вам нужно переписать функциональность, наш вам совет: сделайте новую функцию с новым именем в собственной теме, чтобы избежать конфликтов имен, и настройте файлы шаблонов для ее вызова.

Кроме того, каркасы темы иногда могут быть весьма продвинутыми и включать в себя многочисленные локации произвольных ловушек и фильтров, которые вам могут пригодиться. Используя их, вы можете задействовать `functions.php` дочерней темы для внедрения и изменения HTML родительской темы, не делая вторую копию файлов шаблонов. Этот процесс основывается на темах, охваченных в главе 8. Это куда более сложный способ построения дочерних тем, обеспечивающий, однако, большую безопасность, поскольку в будущем вы сможете обновить родительскую тему, а дочерняя использует ловушки родительской, чтобы изменять тему, превращая ее в индивидуальную.

Как видите, дочерняя тема — еще один мощный инструмент в арсенале WordPress. Вы можете быстро поднять и запустить тему, используя готовую тему как основу, а затем изменить и расширить только что, что вам нужно. Все это потому, что в будущем вам будет обеспечена обновляемость базовой темы.

Рис. 9.5. Дочерняя тема облегчает добавление стиля на определенные страницы

Это в самом деле игра — менять функцию, как только вы ее поняли. Это непростая концепция, особенно в случае использования каркаса с произвольными ловушками и фильтрами. Даже если ваша дизайнерская команда будет кодировать тему с нуля вручную, наличие основы дает множество преимуществ, включая улучшенную производительность, поскольку часто внедряемые возможности уже были внедрены и снабжены знакомым словарем CSS пометок, безусловно, хорошо известным вашей команде. Вне зависимости от того, является ваша родительская тема одним из популярных каркасов тем или чем-то, разработанным непосредственно у вас, преимущества вполне ощутимы.

Темы класса «премиум» и другие каркасы тем

До сих пор в этой главе мы рассматривали тему Twenty Eleven и использовали ее в большинстве примеров. Но это, конечно, не единственная тема или каркас и, возможно, не лучший вариант для вашей команды разработчиков.

Большинство других тем, с которыми вы столкнетесь, обладают другим уровнем адаптивности или набором функций в файле `functions.php`. В целом эта адаптивность дает возможность изменить тему в консоли WordPress. Это может быть идеальным вариантом для некоторых клиентов, не являющихся знатоками PHP и желающих передать контроль администратору сайта, а не разработчикам.

Здесь лучший способ сделать выбор — попробовать каждую тему и подогнать. Найдите тему или каркас, соответствующие вашему стилю кодирования и потребностям, и начните с нее. Помните, что сделать дочернюю тему можно практически из любой темы, равно как и подогнать ее под ваши потребности (в соответствии с лицензией, конечно), но с появлением новой функциональности в WordPress 2.7 произошел скачок в каркасах тем. Сейчас мы пунктиром пройдем по самым популярным (на момент написания) каркасам. Многие останутся неохваченными, так что не забудьте осмотреться сами.

Пожалуйста, не забывайте, что различные термины используются в соответствии с темами. *Magazine themes (журнальные темы)* и *premium themes (темы класса «премиум»)* для разных людей означают разное. Иногда «премиум» означает, что тема стоит денег, иногда — что в теме есть консоль администратора. Некоторые платные темы называются *commercial (коммерческими)*. Каркас темы обычно разрабатывается с расчетом на то, что он будет использоваться как фундамент. Хотя он может работать и сам по себе, он специально пишется для дальнейшего расширения. Темы *starter (стартовые)* делаются для создания ответвления и редактирования на месте. Ваши потребности могут меняться от проекта к проекту, так что вам определенно нужно найти тему, говорящую с вами на одном языке.

Далее приведены примеры некоторых наиболее популярных тем, которые мы использовали в прошлом. Их существует куда больше, и ни одной мы не отдаем преимущества.

Тема Bones

Bones — стартовая тема, сделанная Эдди Мэчадо. В качестве основы для нее используется HTML5, а значит, все было продумано заранее. Нам нравится философия Эдди, сводящаяся к тому, что дочерние темы и каркасы — это круто, но иногда они несколько усложняют ситуацию. Простота возможности взять стартовую тему и сделать из нее тему для единичного сайта проекта улыбается многим сезонным разработчикам и является целью темы Bones.

Помимо HTML5, тема Bones наделена адаптивной базой, использующей медиазапросы, и имеет приоритетом мобильные устройства. Адаптивный веб-дизайн рассматривается несколько подробнее в главе 12, но обратите внимание уже сейчас, что если вы делаете сайты, которые будут просматриваться на смартфонах, наличие адаптивного отображения полезно. Bones также снабжена функциональностью LESS и Sass CSS для продвинутых разработчиков. Взгляните на Bones: http://themble.com/bones.

Тема Carrington

Carrington — это каркас темы, разработанный Crowd Favorite Алекса Кинга. Он спроектирован для дочерних тем и создан быть модульным, то есть в него можно добавить любую функциональность. Carrington делает это путем разбиения кода на мелкие компоненты. Это позволяет использовать код повторно и сохранять порядок. На первый взгляд Carrington может показаться сложной, но как только вы посмотрите, какие процессы вовлечены, все встанет на свои места.

У Crowd Favorite есть и другие интересные проекты WordPress, в том числе RAMP, упоминавшийся в главе 3, и тема Carrington Build, генерирующая разметку с перетаскиванием мышью. Узнать больше о теме Carrington можно тут: http://carringtontheme.com.

Тема Genesis

Каркас тем Genesis, сделанный StudioPress Брайана Гарднера, один из самых, если не *самый*, популярный каркас. Он спроектирован исключительно для дочерних тем, и студия StudioPress даже продает много вариантов. У каркаса Genesis есть множество дополнительных ловушек и фильтров для создания по-настоящему уникальной дочерней темы.

Каркас темы Genesis — результат эволюции темы Revolution. Эта тема была пионером среди тем WordPress и действительно дала зеленый свет для изменения стандартов разработки темы. Revolution была одной из первых тем, охвативших журнальный стиль, что позволило WordPress преодолеть блог-стереотип и стать жизнеспособным CMS-решением. Темы в журнальном стиле заставили WordPress выглядеть менее блогообразным и больше похожим на традиционный веб-сайт. Кроме того, Revolution была одной из первых коммерческих тем. С тех пор она вышла на пенсию и более недоступна. Однако StudioPress воспользовалась наработанным опытом и создала Genesis и множество дочерних тем.

Тема Genesis онлайн: http://studiopress.com.

Тема Hybrid Core

Тема Hybrid от Джастина Тэдлока бесплатная, но взымает членский взнос за доступ к документации по теме, самоучителям и форумам с поддержкой. Снабжена симпатичной консолью для включения и выключения различных возможностей и ловушек CSS. У темы Hybrid есть несколько готовых дочерних тем.

В теме есть множество ловушек CSS внутри записей и тегов body. В ней также много областей для виджетов и шаблонов произвольных страниц. Шаблоны действительно охватывают множество случаев использования и улучшают тему, если вы знаете, как ими пользоваться. Тема Hybrid Core — модульная, обладает большим числом возможностей и расширений, которые можно активировать при необходимости.

Больше информации о теме Hybrid Core вы найдете по ссылке: http://themehybrid.com/.

Roots

Тема Roots, разработанная Беном Уордом, — еще одна стартовая тема. Подобно Bones, она основана на HTML5 и на Twitter Bootstrap. Поскольку тема основана на Twitter Bootstrap, она адаптируется для мобильных устройств.

Одна из интересных особенностей темы Roots — включение в нее файла `.htaccess` для использования на веб-серверах Apache. `.htaccess` очищает URL на сайте, перенаправляя некоторые распознаваемые пути WordPress для изображений, CSS, JavaScript и плагинов так, чтобы они становились более традиционными путями URL. Кроме того, `.htaccess` переписывает поисковые URL так, чтобы они были читаемыми для человека, и делает большинство URL относительными по корневой директории. В целом эта тема делает хорошую работу по дальнейшей очистке URL WordPress и улучшению их семантики.

Тема Roots доступна онлайн: http://www.rootstheme.com/.

Тема StartBox

StartBox — каркас темы, разработанный Брайаном Ричардсом для построения дочерних тем. Подобно теме Hybrid, шаблон является бесплатным, но есть сбор за членство в клубе для доступа к дочерним темам, самоучителям и поддержке.

Тема StartBox включает в себя расширяемую консоль параметров темы и встроенные сокращенные коды для часто запрашиваемых задач, таких как кнопки-действия и ссылки на социальные медиа. Интересной особенностью темы является менеджер боковых панелей, который позволяет администратору сайта создавать зоны виджетов через консоль. Тема StartBox, подобно многим другим шаблонам, имеет своим предком такую уважаемую тему, как Sandbox, то есть в ней много ловушек CSS, встроенных посредством HTML.

Узнать больше о теме StartBox можно по ссылке http://wpstartbox.com/.

Тема Thematic

Thematic — каркас темы, разработанный Яном Стюартом. Тема бесплатная, доступен также целый выводок дочерних тем. Дочерние темы есть как бесплатные, так и коммерческие. Thematic также снабжена минимальной консолью администратора для изменения ограниченного объема информации.

Две особенности действительно являются выдающимися. Во-первых, среди предков этой темы — тема Sandbox. Ловушки CSS с богатой семантикой, имеющиеся в Sandbox, были перенесены в Thematic и расширены. Во-вторых, в этой теме есть 13 областей виджетов. В Thematic много возможностей по оптимизации для

поисковых систем и изменению разметки. Больше информации по этой теме можно найти тут: http://themeshaper.com/thematic/.

Резюме

В этой главе речь шла о том, как использовать темы для организации, структуризации и презентации контента. Тема — лицо сайта. Как бы ни было хорошо содержимое, именно его презентация определяет, насколько доволен будет пользователь. Неумело сделанная тема может подорвать доверие ко всему сайту, тогда как четкая, профессионально построенная тема даст обратный эффект.

В следующей главе мы посмотрим, как использовать внешние источники контента и встраивать их в сайт на WordPress для дальнейшего улучшения качества содержимого и пользовательского опыта.

10 Multisite

В этой главе:

- Понимание WordPress Multisite
- Различия между Multisite и стандартным WordPress
- Установка и конфигурирование сети Multisite
- Работа с Multisite
- Схема базы данных Multisite

Код на Wrox.com для этой главы

Код на wrox.com для этой главы можно скачать по ссылке www.wrox.com/remtitle.cgi?isbn=9781118442272 во вкладке Download Code. Код находится в файле для главы 10 и назван в соответствии с именами файлов кода в этой главе.

WordPress Multisite — мощная возможность ядра в WordPress. Будучи активированным, Multisite позволяет вам создавать несколько веб-сайтов в рамках одной копии WordPress. Это облегчает быстрый запуск новых сайтов на WordPress. Сеть Multisite допускает открытую регистрацию пользователя и сайта, позволяя любому создать новый сайт в сети. Крупнейшая сеть WordPress Multisite — это WordPress.com, прекрасный пример того, чем может быть Multisite.

Что такое Multisite?

Вплоть до версии WordPress 3.0 Multisite назывался WordPress MU (или многопользовательский) и представлял собой отдельный пакет программного обеспечения, который нужно было скачать и установить. В WordPress 3.0 MU был интегрирован в ядро WordPress, так родился WordPress Multisite.

Multisite не активируется по умолчанию, поэтому важно понимать его отличия, перед тем как активировать его в вашей копии WordPress.

Терминология Multisite

Важно понимать терминологию, используемую в этой главе, при работе с WordPress Multisite. Два важных понятия в Multisite — это *сеть (network)* и *сайт(site)*. Сеть — это вся копия Multisite целиком. Сайт — отдельный сайт внутри этой сети. То есть WordPress Multisite представляет собой сеть сайтов.

Другое важное понятие — *идентификатор блога (Blog ID)*. ID блога — это уникальный идентификатор, приписываемый каждому сайту, создаваемому в Multisite. Многие из примеров функций и кодов будут ссылаться на ID блога. Иногда его также называют *идентификатором сайта (site ID)*. Не забывайте, что WordPress изначально был платформой для блогов, но с годами развился в полноценную систему управления контентом. Поэтому во многих функциях и в коде WordPress элементы по-прежнему обозначаются как «блоги», хотя на самом деле они являются сайтами.

ЗАМЕЧАНИЕ

Пусть термин «блог» вас не смущает. Блог в WordPress Multisite — это просто сайт в Сети. ID блога, на который ссылается множество функций и примеров кода, — это всего лишь уникальный идентификатор сайта в Сети.

Отличия

Когда вы устанавливаете стандартный WordPress, вы устанавливаете одиночный веб-сайт, работающий на WordPress. WordPress Multisite позволяет запускать неограниченное число веб-сайтов в одной копии WordPress. При активировании Multisite нужно определить, как сайты будут отображаться в WordPress: с помощью поддоменов или поддиректорий. Вот пример обоих вариантов:

Поддиректория

- http://example.com/site1
- http://example.com/site2

Поддомен

- http://site1.example.com/
- http://site2.example.com/

Как видите, это важное решение, над которым нужно хорошенько поразмыслить. С помощью плагина можно приписать домены верхнего уровня любому сайту в Сети (то есть http://mywebsite.com), о чем речь пойдет далее.

Работа с темами и плагинами в Multisite также осуществляется по-другому. Администраторы отдельных сайтов могут активировать плагины и темы для своего сайта, но не могут их устанавливать.

WordPress Multisite также вводит новую пользовательскую роль: Super Admin (суперадминистратор). У пользователей-суперадминистраторов есть доступ к разделу администрирования сети Multisite. Именно там конфигурируется Multisite. У суперадминистраторов также есть полный доступ к каждому сайту в сети Multisite, тогда как обычным администраторам доступен только тот сайт, который они администрируют.

Преимущества Multisite

У WordPress Multisite есть ряд преимуществ перед стандартным WordPress. Самое большое — наличие единственной копии WordPress для администрирования. Это существенно упрощает обновление WordPress, тем и плагинов. Если у вас есть сеть WordPress Multisite с 50 сайтами и обновление для плагина, вам нужно установить его только один раз, после чего оно будет действительно для всех сайтов в сети. Если же каждый из 50 сайтов представляет собой отдельную копию WordPress, придется обновлять плагин 50 раз.

Собранный воедино контент — еще одно большое преимущество. Совсем несложно передавать содержимое между сайтами сети Multisite. Например, если в сети 50 сайтов, вы можете с легкостью передавать записи из каждого сайта в основной блог, чтобы демонстрировать свою сеть сайтов и ее контент.

Самое большое преимущество при использовании Multisite — это скорость, с которой вы можете запускать новые сайты. Всего несколькими щелчками мыши вы можете создавать новые сайты в сети. Эти сайты могут иметь общие темы, плагины и даже пользователей.

Активация Multisite

Активация возможности Multisite в WordPress — довольно понятный процесс. Первый шаг: добавить следующую строку кода в файл `wp-config.php`:

```
define( 'WP_ALLOW_MULTISITE', true );
```

Ее нужно вставить прямо над этим комментарием:

```
/* Это всё, дальше не редактируем. Успехов! */
/* That's all, stop editing! Happy blogging. */
```

Сохраните файл `wp-config.php` и загрузите на сервер. Теперь войдите в консоль администратора WordPress, и вы увидите новую вкладку в меню **Инструменты ▸ Сеть**, как показано на рис. 10.1.

Экран Сеть будет варьироваться в зависимости от текущей установки WordPress. Если она позволяет, вы сможете выбрать поддомены или поддиректории для установки Multisite. Вам также необходимо проверить правильность значений адреса сервера, названия сети и электронного адреса администратора. Эти поля WordPress заполняет автоматически, но при необходимости значения можно изменить. После того как вы подтвердили, что значения в порядке, нажмите на кнопку Установить, чтобы установить Multisite в WordPress.

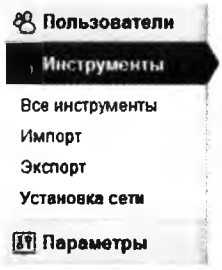


Рис. 10.1. Подпункт меню Сеть

Последний шаг для активации Multisite будет представлен на экране; это серия изменений, которые необходимо внести в WordPress вручную. Первый шаг: создание директории `blogs.dir` в директории `/wp-content`. Здесь будут храниться все медиафайлы, загруженные через сеть Multisite. Все они попадают в `wp-content/blogs.dir/BLOG_ID/files/YEAR/MONTH`. Постоянная ссылка на медиафайл выглядит, например, так: http://example.com/files/2013/10/Halloween.png.

Следующий шаг — добавление определенного кода в файл `wp-config.php`. Этот код определяет базовые настройки для Multisite и будет варьироваться в зависимости от установки. Ниже приведен пример кода для варианта Multisite с поддиректориями с доменом `example.com`.

```
define( 'MULTISITE', true );
define( 'SUBDOMAIN_INSTALL', false );
$base = '/';
define( 'DOMAIN_CURRENT_SITE', 'example.com' );
define( 'PATH_CURRENT_SITE', '/' );
define( 'SITE_ID_CURRENT_SITE', 1 );
define( 'BLOG_ID_CURRENT_SITE', 1 );
```

Последний шаг — замена правил файла `.htaccess` на новые:

```
RewriteEngine On
RewriteBase /
RewriteRule ^index\.php$ - [L]
# uploaded files
RewriteRule ^([_0-9a-zA-Z-]+/)?files/(.+) wp-includes/ms-files.php?file=$2 [L]
# add a trailing slash to /wp-admin
RewriteRule ^([_0-9a-zA-Z-]+/)?wp-admin$ $1wp-admin/ [R=301,L]
RewriteCond %{REQUEST_FILENAME} -f [OR]
```

```
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^ - [L]
RewriteRule  ^[_0-9a-zA-Z-]+/(wp-(content|admin|includes).*) $1 [L]
RewriteRule  ^[_0-9a-zA-Z-]+/(.*\.php)$ $1 [L]
RewriteRule . index.php [L]
```

Эти правила могут разниться в зависимости от установки Multisite, поэтому убедитесь, что вы копируете код, представленный на экране Сеть при активации Multisite в WordPress.

Сделав требуемые изменения, вам нужно будет задокументировать их в WordPress. Самый простой способ указать время активации Multisite — через консоль WordPress. Вы увидите новый элемент меню Мои сайты. Первым подпунктом в нем будет Администратор сети, то есть главный администратор консоли Multisite. В меню Мои сайты также будут перечислены все сайты в сети, членом которых вы являетесь, как показано на рис. 10.2.

WordPress Multisite установлен и готов к использованию!

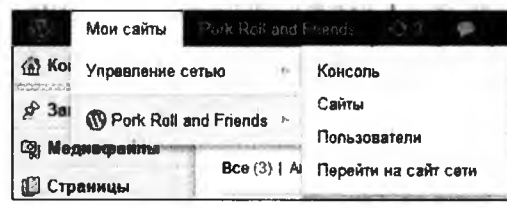


Рис. 10.2. Меню Сеть в Multisite

Работа в сети

Теперь, когда вы знаете, как активировать Multisite в WordPress, важно понять, как управлять сетью. Здесь рассматривается раздел администратора сети Multisite WordPress и управление сетью.

Консоль администратора сети

Консоль администратора сети WordPress Multisite — центральный пост управления всей сетью. Вы можете попасть туда через меню Мои сайты в консоли WordPress или по ссылке http://example.com/wp-admin/network/.

Консоль администратора сети кажется знакомой, поскольку ее разметка схожа с разметкой стандартной консоли WordPress.

Создание сайтов и управление ими

Чтобы увидеть список всех сайтов в сети Multisite, зайдите в меню Сайты. Здесь отображается каждый сайт, зарегистрированный в Multisite, вне зависимости от

его статуса. Список снабжает вас некоторой важной информацией, такой как имя сайта, дата последнего обновления, дата регистрации и список всех пользователей, являющихся членами сайта. Чтобы отредактировать настройки сайта, щелкните на его имени, это вызовет экран редактирования.

Раздел редактирования сайта позволяет редактировать все настройки определенного сайта в сети. Вы увидите вкладки для каждого раздела настроек, как показано на рис. 10.3.

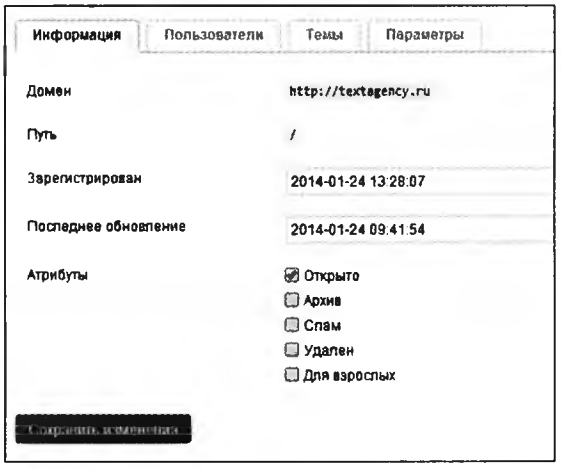


Рис. 10.3. Раздел редактирования сайта

Вкладка **Информация** позволяет редактировать домен и путь к сайту. Эти две настройки недоступны для главного сайта в сети. Вы можете также изменить дату регистрации и последнего обновления и метки времени. Последний редактируемый элемент — атрибуты или настройки статуса сайта. У каждого сайта в сети может быть один из следующих пяти статусов:

1. `Public` — сайт доступен публично, если настройки приватности установлены на доступность для поисковых систем.
2. `Archived` — сайт архивирован и больше недоступен публично.
3. `Spam` — сайт был сочтен спамом и недоступен публично.
4. `Deleted` — сайт отмечен для удаления и недоступен публично.
5. `Mature` — сайт помечен как только для взрослых.

Только два статуса не изымают сайт из публичного доступа: `Public` и `Mature`.

Вкладка **Пользователи** позволяет вам управлять тем, у каких пользователей есть доступ к сайту. Здесь перечисляются все пользователи с их ролями. Вы также можете добавлять на сайт новых пользователей. В следующем разделе пользователи Multisite рассматриваются более подробно.

Вкладка Темы позволяет вам активировать или деактивировать темы для сайта. Активация темы здесь не означает ее активации для редактируемого сайта, она просто становится доступной для администратора сайта, который сможет ее активировать, если сочтет это нужным.

Вкладка Настройки позволяет вам редактировать другие настройки сайта. Здесь есть множество параметров. Но если вы не знаете, что именно меняете, лучше этого не делать.

Работа с пользователями и ролями

Пользователи в сети Multisite не то же, что пользователи в стандартном WordPress. Основное отличие заключается в том, что у каждого сайта в сети может быть свой набор пользователей. Пользователи также могут быть членами нескольких сайтов в сети и даже иметь разные роли для каждого сайта. Например, вы можете быть администратором на сайте A, но всего лишь автором на сайте X.

Если в меню Сеть активировано разрешение новой регистрации, посетители могут создавать новые учетные записи в WordPress. Новый пользователь автоматически становится не членом всех сайтов в сети, а только членом главного (первого) сайта. Например, если в вашей сети два сайта: Halloween и Christmas, любой регистрирующийся посетитель становится членом сайта Halloween, но не Christmas. Конечно, вы всегда можете добавить этого пользователя на сайт Christmas позднее.

Чтобы просмотреть всех пользователей в сети Multisite, зайдите в меню Пользователи. Здесь вы увидите список всех пользователей в сети, включая их имена, адреса электронной почты, даты регистрации и список сайтов, членами которых они являются. Если пользователь является суперадминистратором, вы увидите информацию рядом с его именем пользователя. Вы можете с легкостью добавлять, редактировать или удалять пользователей из сети в данной вкладке.

Темы и плагины

Multisite обращается с темами и плагинами иначе, нежели стандартный WordPress. Все сайты в сети могут пользоваться одинаковыми плагинами или темами либо совершенно разными их наборами. Такая гибкость демонстрирует мощь Multisite в WordPress.

Темы

Чтобы увидеть все темы, установленные в WordPress, зайдите во вкладку меню Темы. Во вкладке тем раздела администратора сети есть список всех тем, похожий на список в обычном WordPress. Основное отличие в том, что вместо ссылки `Activate` для каждой темы вы увидите ссылку `Network Enable`. Активация для сети любой темы из списка сделает тему доступным параметром для всех сайтов в сети. Это не означает, что тема активируется, она просто становится доступной для

администраторов сайта во вкладке **Внешний вид ▸ Темы в WordPress**. Это позволит вам контролировать, какие темы доступны для выбора администраторами сайтов.

Плагины

Плагины в Multisite работают не так, как темы. Они могут быть активированы для сети, это означает, что плагин будет запущен на каждом сайте в сети. Если плагин не активирован для сети, он по-прежнему может быть активирован на уровне сайта. То есть вы можете запускать плагины на любом или на всех сайтах в сети.

Чтобы просмотреть все плагины, доступные для использования, зайдите во вкладку меню **Плагины**. Здесь вы увидите список плагинов, скачиваемых для WordPress. Нажатие на **Активировать для сети** активирует плагин для каждого сайта в сети. Если плагин для сети не активирован, его можно будет активировать на уровне сайта в стандартной вкладке меню **Плагины**.

ЗАМЕЧАНИЕ

Важно убедиться, что плагин совместим с Multisite, до того как активировать его для сети. Если плагин не был проверен на Multisite или кодирован должным образом, есть вероятность, что он при активации для сети сломает сеть.

Настройки

Меню настроек позволяет установить в Multisite настройки, распространяющиеся на всю сеть. Здесь вы можете активировать для пользователей регистрацию учетной записи и даже создание сайта. Активация обеих возможностей позволит посетителям регистрировать учетные записи и даже создавать новые сайты в сети.

Вы также можете определить, какие типы файлов разрешены для загрузки, общий размер места для хранения загруженных файлов и даже максимальный размер отдельного файла. Если вы планируете запуск очень большой сети Multisite, ограничения пространства для загрузки и максимального размера файла поможет вам сохранить значительное дисковое пространство.

По умолчанию меню плагинов скрыто от администраторов сайтов. Только суперадминистраторам позволяется активировать и деактивировать плагины на уровне сайта. Если вы хотите активировать меню плагинов для обычных администраторов, это можно сделать во вкладке настроек меню.

Привязка домена

Распространенной возможностью для пользователей Multisite является привязка домена. Ранее мы говорили о двух предустановленных конфигурациях для Multisite: поддиректория или поддомен. Но что, если вы хотите, чтобы у каждого сайта в сети было уникальное доменное имя? Для этого есть плагин! WordPress MU Domain

Mapping (http://wordpress.org/extend/plugins/wordpress-mu-domain-mapping/) позволяет это сделать. Этот плагин упрощает прикрепление домена верхнего уровня к любому сайту в сети. Он работает и с поддиректориями, и с поддоменами. Например, вместо двух сайтов типа http://example.com/brad и http://example.com/myers вы можете сделать http://brad.com и http://myers.com. Все URL будут с использованием домена верхнего уровня, который вы приписали сайтам.

Кодирование для Multisite

Когда Multisite активирован в WordPress, вы можете использовать преимущества целого нового набора особых функций и API в темах и плагинах. Понимание новых возможностей поможет вам сделать темы и плагины совместимыми с Multisite.

Идентификатор блога

У каждого сайта в сети Multisite есть уникальный идентификатор. Он называется идентификатором блога (Blog ID). Почти каждая функция, с которой вы работаете при написании кода для Multisite, будет его использовать. Именно так WordPress определяет, с каким сайтом вы хотите работать.

Идентификатор блога также используется в префиксах таблиц базы данных, которые каждый новый сайт создает в сети. При построении новых сайтов в Multisite WordPress создает дополнительные таблицы баз данных для хранения контента и настроек сайта. Например, если вы создаете сайт в сети, WordPress создаст новые таблицы с префиксами типа `wp_2_posts`, где `wp_` — префикс таблицы, который вы определили при установке WordPress, а `2_` — идентификатор нового сайта.

Идентификатор блога хранится в глобальной переменной `$blog_id`:

```
<?php
global $blog_id;
echo 'Current Blog ID: ' .$blog_id;
?>
```

В Multisite `$blog_id` всегда будет ID сайта, который вы просматриваете в данный момент. В стандартном WordPress глобальная переменная `$blog_id` всегда будет 1.

Общие функции

При работе с Multisite доступны общие функции. Самая важная функция при работе с Multisite — это `is_multisite()`:

```
<?php
if ( is_multisite() ) {
    echo 'Multisite is enabled';
}
?>
```

Она определяет, активирован ли Multisite, и в случае положительного ответа возвращает `true`. Каждый раз, когда вы собираетесь использовать функции для Multisite в WordPress, важно убедиться, что Multisite активирован. Если он не активирован, а функция запущена, вы получите сообщение об ошибке и сломаете сайт.

Еще одна полезная функция возвращает информацию о сайте: `get_blog_details()`.

```
<?php get_blog_details( $fields, $getall ); ?>
```

Она принимает два параметра:

1. `$fields` — Blog ID, имя блога или массив полей для запроса.
2. `$getall` — запрос подробностей.

Используя эту функцию, можно получить общую информацию по любому сайту:

```
<?php print_r( get_blog_details( 1 ) ); ?>
```

Запуск показанного ранее примера кода даст следующие результаты:

```
stdClass Object
(
    [blog_id] => 1
    [site_id] => 1
    [domain] => example.com
    [path] => /
    [registered] => 2012-10-31 19:01:47
    [last_updated] => 2012-10-31 19:01:49
    [public] => 1
    [archived] => 0
    [mature] => 0
    [spam] => 0
    [deleted] => 0
    [lang_id] => 0
    [blogname] => Halloween Site
    [siteurl] => http://example.com
    [post_count] => 420
)
```

Переключение и восстановление сайтов

Одним из основных преимуществ использования WordPress Multisite является легкость обмена контентом и другими данными между разными сайтами в сети.

Есть две основные функции, которые можно использовать для возврата данных с других сайтов в сети. Первая функция — это `switch_to_blog()`. Она позволяет переключиться на любой сайт в сети:

```
<?php switch_to_blog( $blog_id, $validate ); ?>
```

Функция принимает два параметра:

1. `$blog_id` — ID сайта, на который вы хотите переключиться.
2. `$validate` — проверяет, существует ли сайт, перед обработкой. Значение по умолчанию `false`.

Вторая функция — это `restore_current_blog()`. Она делает именно то, о чем говорит название: восстанавливает предыдущий сайт после вызова функции `switch_to_blog()`. Для этой функции нет параметров, вы создаете произвольный сокращенный код для возврата последних пяти записей из сайта в сети:

```
add_shortcode( 'show_network_posts', 'prowp_get_network_posts' );
```

Сначала зарегистрируйте новый сокращенный код под названием `show_network_posts`, используя функцию `add_shortcode()`. Новый сокращенный код будет принимать один параметр: Blog ID сайта, с которого будут отображаться последние записи. Затем вы создаете функцию, возвращающую последние записи с сайта в сети.

```
function prowp_get_network_posts( $atts ) {
     extract( shortcode_atts( array(
          'blog_id' => '1'
     ), $atts ) );

    // проверяем, включен ли Multisite
    if ( is_multisite() ) {

        // переключаемся на блог с переданным ID
        switch_to_blog( absint( $blog_id ) );

        // создаем произвольный цикл
        $recent_posts = new WP_Query();
        $recent_posts->query( 'posts_per_page=5' );
        $site_posts = '';

        // начинается произвольный цикл
        while ( $recent_posts->have_posts() ) :
            $recent_posts->the_post();

            // сохраняем последние записи в переменной
            $site_posts .= '<p><a href="'.get_permalink().'">'
                .get_the_title().'</a></p>';

        endwhile;

        // возвращаемся к текущему сайту
        restore_current_blog();

    }

    // возвращаем записи
    return $site_posts;

}
```

Как всегда, вам нужно убедиться, что Multisite активирован, используя функцию проверки `is_multisite()`. Затем используйте функцию `switch_to_blog()`, чтобы переключиться на сайт, Blog ID передается через сокращенный код. Если Blog ID в сокращенном коде задан не был, по умолчанию он будет установлен на Blog ID 1. Теперь, когда вы переключились на сайт, вам нужно создать произвольный цикл,

используя `WP_Query`, чтобы извлечь последние записи из сайта. Затем нужно прогнать через цикл результаты `WP_Query`, сохраняя каждую запись в переменную с названием `$site_posts`.

По окончании цикла нужно запустить `restore_current_blog()`, чтобы переключиться на предыдущий сайт, который вы просматривали. Если вы не запустите эту функцию, WordPress останется на Blog ID 10, поэтому все подчиненные циклы или произвольный код будут считать, что вы там, хотя на самом деле это не так. Завершающий шаг — возвращение переменной `$site_posts`, содержащей последние пять записей из Blog ID 10.

Готово! Теперь вы можете с легкостью отображать последние записи любого сайта в вашей сети, используя сокращенный код: `[show_network_posts blog_id="10"]`. В листинге 10.1 показан полный код, выполненный как плагин.

Листинг 10.1. Пример сокращенного кода (prowp2-multisite-shortcode.zip)

```
<?php
/*
Plugin Name: ProWP2 Multisite Shortcode Example
Plugin URI: http://strangework.com/wordpress-plugins
Description: A shortcode to display posts from any site in your network
Version: 1.0
Author: Brad Williams
Author URI: http://strangework.com
License: GPLv2
*/
add_shortcode( 'show_network_posts', 'prowp_get_network_posts' );
function prowp_get_network_posts( $atts ) {
     extract( shortcode_atts( array(
          'blog_id' => '1'
     ), $atts ) );

    // проверяем, включен ли Multisite
    if ( is_multisite() ) {

        // переключаемся на блог с переданным ID
        switch_to_blog( absint( $blog_id ) );

        // создаем произвольный цикл
        $recent_posts = new WP_Query();
        $recent_posts->query( 'posts_per_page=5' );
        $site_posts = '';

        // начинается произвольный цикл
        while ( $recent_posts->have_posts() ) :
            $recent_posts->the_post();

            // сохраняем последние записи в переменной
            $site_posts .= '<p><a href="'.get_permalink().'">'
                .get_the_title().'</a></p>';

        endwhile;
```

```
        // возвращаемся к текущему сайту
        restore_current_blog();

    }

    // возвращаем записи
    return $site_posts;

}
?>
```

Функция `switch_to_blog()` не ограничивается контентом сайта, но также может возвращать и другие данные WordPress, включая меню, виджеты, боковые панели и многое другое. Обычно любые данные, которые хранятся в таблицах базы данных контента (`wp_blogid_tablename`), доступны при использовании `switch_to_blog()`.

Рассмотрим другой пример. На этот раз возвращается определенное меню с сайта в сети Multisite.

```
<?php
switch_to_blog( 10 );
wp_nav_menu( 'Main Menu' );
restore_current_blog();
?>
```

Сначала запустите `switch_to_blog()`, чтобы переключиться на Blog ID 10. Затем используйте функцию WordPress `wp_nav_menu()` WordPress, чтобы отобразить меню с названием Main Menu с этого сайта. Наконец, запустите `restore_current_blog()`, чтобы вернуться к текущему сайту, который вы просматриваете. Приведенный выше код отобразит навигационное меню `Main Menu` сайта с ID 10 там, где будет запущен код.

Важно обратить внимание на то, что функция `switch_to_blog()` способна генерировать очень большие запросы SQL, которые могут вызвать проблемы с производительностью WordPress. Лучше кэшировать любые данные, используемые этой функцией, в переходном процессе, что позволит временно сохранить их в кэшированной версии. Переходные процессы WordPress рассматриваются в главе 11.

Еще одно важное замечание: `switch_to_blog()` меняет только контекст базы данных, не перенимая всю конфигурацию сайта, то есть плагины при переключении не передаются. Если вы переключаетесь на сайт и пытаетесь выполнить функцию, определенную для плагина, который не активирован, вы получите сообщение об ошибке.

Создание нового сайта

Вы научились создавать новые сайты через консоль администратора сети Multisite, теперь посмотрим, как создавать их с помощью кода. Чтобы сделать это, воспользуемся функцией `wpmu_create_blog()`.

```
<?php wpmu_create_blog($domain, $path, $title, $user_id, $meta, $site_id); ?>
```

Функция принимает шесть параметров:

1. `$domain` — домен нового сайта.
2. `$path` — путь к новому сайту.
3. `$title` — название нового сайта.
4. `$user_id` — ID аккаунта пользователя, который будет администратором сайта.
5. `$meta` — дополнительная метаинформация.
6. `$site_id` — ID создаваемого сайта.

Обязательными являются только первые четыре параметра, оставшиеся два — дополнительные. Параметр `$site_id` используется, только если вы собираетесь запускать несколько сетей внутри одной копии WordPress. Если новый сайт создан успешно, функция вернет его новый Blog ID.

Например, вы можете построить плагин, который использует только функцию `wpmu_create_blog()` для создания новых сайтов в сети Multisite:

```
add_action( 'admin_menu', 'prowp_multisite_create_menu' );
function prowp_multisite_create_menu() {
    // создаем произвольное меню верхнего уровня
    add_menu_page( 'Multisite Create Site Page', 'Multisite Create Site',
        'manage_options', 'prowp-network-create', 'prowp_multisite_create_sites' );
}
```

Сначала вы создаете новое меню верхнего уровня с названием Multisite Create Site. Это меню будет соединяться с произвольной функцией `prowp_multisite_create_sites()`, что позволит вам создавать сайты в сети. Продолжим и создадим эту функцию следующим образом:

```
function prowp_multisite_create_sites() {
    // проверяем, включен ли Multisite
    if ( is_multisite() ) {
```

Не забывайте всегда проверять, активирован ли Multisite, используя функцию `is_multisite()`. Затем добавьте код для возвращения значений полей отправляемой формы и создайте новый сайт в Multisite:

```
// обрабатываем форму, если она отправлена
if ( isset( $_POST['create_site'] ) ) {
    // устанавливаем переменные значения из формы
    $domain = strip_tags( $_POST['domain'] );
    $path = strip_tags( $_POST['path'] );
    $title = strip_tags( $_POST['title'] );
    $user_id = absint( $_POST['user_id'] );
    // проверяем, установлены ли необходимые значения
    if ( $domain && $path && $title && $user_id ) {
        // создаем новый сайт в сети
        $new_site = wpmu_create_blog( $domain, $path, $title, $user_id );
        // в случае успеха сообщаем об этом
        if ( $new_site ) {
            echo '<div class="updated">New site ' .$new_site
```

```
                . ' created successfully!</div>';
        }
    // если необходимые значения не установлены, отображаем сообщение об ошибке
    } else {
        echo '<div class="error">New site could not be created.
            Required fields are missing</div>';
    }
}
```

Сначала проверьте, установлен ли параметр `$_POST['create_site']`. Это поле будет задано, только если форма была отправлена. Затем вы заполняете значения переменных данными, отправленными через форму. Убедитесь, что вы используете надлежащие функции очистки, чтобы быть уверенными, что отправляемые значения не содержат кодов HTML и PHP. Вы также проверяете, что значение `user_id` является положительным целым, с помощью функции WordPress `absint()`.

Теперь, когда переменные заданы, нужно убедиться, что у каждой есть значение. Если не хватает данных для какого-либо из четырех обязательных параметров, появится сообщение об ошибке. После того как вы убедились, что значения существуют, время выполнить функцию `wpmu_create_blog()`, чтобы создать новый сайт. Если новый сайт создан успешно, переменная `$new_site` будет содержать новый Blog ID сайта.

Теперь построим форму для полей нового сайта:

```
<div class="wrap">
    <h2>Create New Site</h2>
    <form method="post">
    <table class="form-table">
    <tr valign="top">
        <th scope="row"><label for="fname">Domain</label></th>
        <td><input maxlength="45" size="25" name="domain"
            value="<?php echo esc_attr( DOMAIN_CURRENT_SITE ); ?>" /></td>
    </tr>
    <tr valign="top">
        <th scope="row"><label for="fname">Path</label></th>
        <td><input maxlength="45" size="10" name="path" /></td>
    </tr>
    <tr valign="top">
        <th scope="row"><label for="fname">Title</label></th>
        <td><input maxlength="45" size="25" name="title" /></td>
    </tr>
    <tr valign="top">
        <th scope="row"><label for="fname">User ID</label></th>
        <td><input maxlength="45" size="3" name="user_id" /></td>
    </tr>
    <tr valign="top">
        <td>
        <input type="submit" name="create_site"
            value="Create Site" class="button-primary" />
        <input type="submit" name="reset" value="Reset" class="button-secondary" />
        </td>
    </tr>
    </table>
    </form>
</div>
```

Это стандартный HTML для сбора данных из обязательных полей для создания нового сайта в Multisite. Теперь вы можете легко создавать новые сайты в сети, используя плагин, показанный в листинге 10.2.

Листинг 10.2. Пример создания сайтов в Multisite (prowp2-multisite-create-site.zip)

```
<?php
/*
Plugin Name: ProWP2 Create Site Example Plugin
Plugin URI: http://strangework.com/wordpress-plugins
Description: Плагин, демонстрирующий создание сайтов в режиме Multisite
Version: 1.0
Author: Brad Williams
Author URI: http://strangework.com
License: GPLv2
*/
add_action( 'admin_menu', 'prowp_multisite_create_menu' );
function prowp_multisite_create_menu() {
    // создаем произвольное меню верхнего уровня
    add_menu_page( 'Multisite Create Site Page',
        'Multisite Create Site', 'manage_options',
        'prowp-network-create', 'prowp_multisite_create_sites' );
}
function prowp_multisite_create_sites() {

    // проверяем, включен ли Multisite
    if ( is_multisite() ) {

        // обрабатываем форму, если она отправлена
        if ( isset( $_POST['create_site'] ) ) {

            // устанавливаем переменные значения из формы
            $domain = strip_tags( $_POST['domain'] );
            $path = strip_tags( $_POST['path'] );
            $title = strip_tags( $_POST['title'] );
            $user_id = absint( $_POST['user_id'] );

            // проверяем, установлены ли необходимые значения
            if ( $domain && $path && $title && $user_id ) {

                // создаем новый сайт в сети
                $new_site = wpmu_create_blog( $domain, $path,
                    $title, $user_id );

                // в случае успеха сообщаем об этом
                if ( $new_site ) {

                    echo '<div class="updated">New site '
                        .$new_site. ' created successfully!</div>';

                }

            // если необходимые значения не установлены,
            // отображаем сообщение об ошибке
            } else {
```

```
                echo '<div class="error">New site could not be created.
                      Required fields are missing</div>';

            }

        }
        ?>
        <div class="wrap">
            <h2>Create New Site</h2>
            <form method="post">
            <table class="form-table">
            <tr valign="top">
                <th scope="row"><label for="fname">Domain</label></th>
                <td><input maxlength="45" size="25" name="domain"
                    value="<?php echo esc_attr( DOMAIN_CURRENT_SITE ); ?>" /></td>
            </tr>
            <tr valign="top">
                <th scope="row"><label for="fname">Path</label></th>
                <td><input maxlength="45" size="10" name="path" /></td>
            </tr>
            <tr valign="top">
                <th scope="row"><label for="fname">Title</label></th>
                <td><input maxlength="45" size="25" name="title" /></td>
            </tr>
            <tr valign="top">
                <th scope="row"><label for="fname">User ID</label></th>
                <td><input maxlength="45" size="3" name="user_id" /></td>
            </tr>
            <tr valign="top">
                <td>
                <input type="submit" name="create_site"
                    value="Create Site" class="button-primary" />
                <input type="submit" name="reset"
                    value="Reset" class="button-secondary" />
                </td>
            </tr>
            </table>
            </form>
        </div>
        <?php
    } else {

        echo '<p>Multisite is not enabled</p>';

    }

}
?>
```

Меню администратора сети

Ранее в этой главе мы говорили о разделе администратора сети Multisite. В этой консоли конфигурируются все настройки сети. Как и в стандартном WordPress,

в консоль Network Admin можно добавлять пункты и подпункты. Это делается с помощью ловушки `network_admin_menu`:

```
add_action( 'network_admin_menu', 'prowp_add_network_settings_menu' );
```

Ловушка `network_admin_menu` активируется после того, как структура меню администратора страницы оказывается на месте. Второй параметр — это произвольная функция `prowp_add_network_settings_menu()`, которая зарегистрирует новое меню.

```
function prowp_add_network_settings_menu() {
    // добавляем меню настроек
    add_menu_page( 'ProWP2 Options Page', 'ProWP2 Options',
        'manage_options', 'prowp-network-settings', 'prowp_network_settings' );
}
```

Как видите, регистрация нового меню — то же самое, что и регистрация стандартного меню WordPress. В данном примере для создания меню верхнего уровня в консоли администратора сети используется функция `add_menu_page()`, как показано на рис. 10.4.

С той же легкостью, с какой вы добавляете меню верхнего уровня, можно добавить и подпункты меню к уже существующим пунктам. Для этого используйте функцию `add_submenu_page()`:

```
function prowp_add_network_settings_menu() {
    // добавляем подпункты меню
    add_submenu_page( 'settings.php', 'ProWP2 Options Page',
        'ProWP2 Options', 'manage_options', 'prowp-network-settings',
            'prowp_network_settings' );
}
```

Эта функция работает точно так же, как в стандартном WordPress. Первый параметр, в данном случае `settings.php`, — самый важный. Это значение сообщает функции, к какому пункту меню добавлять подпункт. В данном примере к пункту **Настройки** в Network Admin добавляется подпункт с названием **ProWP2 Options**.

Рис. 10.4. Меню верхнего уровня Network Admin

Подпункты можно добавлять к следующим файлам:

- `index.php` — добавление подпункта в **Консоль**;
- `sites.php` — добавление подпункта в **Сайты**;
- `users.php` — добавление подпункта в **Пользователи**;
- `themes.php` — добавление подпункта в **Темы**;
- `plugins.php` — добавление подпункта в **Плагины**;
- `settings.php` — добавление подпункта в **Настройки**;
- `update-core.php` — добавление подпункта в **Обновления**.

Как правило, лучше добавлять настройки сети как подпункт к настройкам в консоли администратора сети. Именно там большинство пользователей станут искать настройки плагинов, как и в стандартном WordPress.

Параметры Multisite

При хранении параметров Multisite важно использовать правильные функции в правильном месте. Вопрос, который вам следует себе задать: нужно ли вам контролировать значения настроек? Если настройка является специфичной для каждого сайта в сети и может варьироваться от сайта к сайту, то параметры следует хранить как настройки сайта. Если настройка будет распространяться на всю сеть и не различаться между сайтами, то ее можно хранить как параметр сети.

Параметры сайта

Чтобы сохранить специфичные для сайта параметры в Multisite, вы можете использовать функции `*_blog_option()`:

- `add_blog_option()` — создать новый параметр.
- `update_blog_option()` — обновить параметр или создать его, если он не существует.
- `get_blog_option()` — загрузить параметр сайта.
- `delete_blog_option()` — удалить параметр сайта.

Эти функции работают практически идентично стандартным функциям параметров WordPress; основное различие в том, что они требуют задать параметр `$blog_id`:

```
<?php add_blog_option( $blog_id, $key, $value ); ?>
```

Параметр `$key` — это имя, которое вы хотите дать параметру, а параметр `$value` — его значение.

Вернуть параметр сайта тоже просто. Приведенный далее пример показывает, как использовать `get_blog_option()`, чтобы вернуть параметры, специфичные для Blog ID 10:

```
<?php
$blog_id = 10;
echo '<p>Site ID: ' .$blog_id .'</p>';
echo '<p>Site Name: ' .get_blog_option( $blog_id, 'blogname' ) .'</p>';
echo '<p>Site URL: ' .get_blog_option( $blog_id, 'siteurl' ) .'</p>';
?>
```

Параметры сети

Для хранения в сети Multisite параметров сети можно использовать функции `*_site_option()`:

- `add_site_option()` — создать новый параметр.
- `update_site_option()` — обновить параметр или создать его, если он не существует.
- `get_site_option()` — загрузить параметр сети.
- `delete_site_option()` — удалить параметр сети.

Эти функции работают практически идентично стандартным функциям параметров WordPress, но их значения хранятся в таблице `wp_sitemeta` базы данных Multisite. Вы можете использовать эти функции для хранения глобальных настроек Multisite, одинаковых для всех файлов в сети.

```
<?php add_site_option( $key, $value ); ?>
```

Обратите внимание, что при добавлении сетевого параметра нет необходимости определять Blog ID. Поскольку вы сохраняете параметр сети, неважно, с какого Blog ID будет выполняться код.

Если Multisite не активирован, а код вызывает одну из функций `*_site_option()`, WordPress вернется к использованию стандартных функций `*_option()`, таких как `add_option()`.

Пример параметров сети

Теперь, когда вы понимаете, как создать и вернуть параметры сети, давайте построим простой плагин параметров сети для Multisite. В рамках этого примера вы создадите плагин для хранения параметров, применяемых по всей сети. Плагин будет иметь способность к откату, так что если пользователь не использует Multisite, параметры будут сохранены для стандартного WordPress.

Первый шаг — добавление меню сетевых настроек.

```
add_action( 'init', 'prowp_network_settings_menu' );
function prowp_network_settings_menu() {

    if ( is_multisite() ) {

        // Multisite включен, поэтому добавляем меню в Network Admin
        add_action( 'network_admin_menu', 'prowp_add_network_settings_menu' );
    } else {

        // Multisite не включен, поэтому добавляем меню
        // в панель администрирования WordPress
        add_action( 'admin_menu', 'prowp_add_network_settings_menu' );
    }
}
```

Для вызова произвольной функции для регистрации меню параметров сети применяется зацепка-действие `init`. Обратите внимание, как в предыдущем примере использована функция `is_multisite()`. Если у пользователя активирован Multisite, новое меню будет добавлено в Network Admin, если нет — меню будет добавлено

как стандартное меню WordPress. Этот код важен для сохранения совместимости, вне зависимости от того, активирован ли Multisite.

Теперь, когда вы зарегистрировали нужную зацепку-действие для меню, необходимо создать произвольную функцию для регистрации нового меню.

```
function prowp_add_network_settings_menu() {
    // добавляем меню настроек
    add_menu_page( 'Network Options Page', 'Network Options',
        'manage_options', 'prowp-network-settings', 'prowp_network_settings' );
}
```

Приведенный код использует функцию `add_menu_page()`, чтобы создать новое меню верхнего уровня, маркированное как Параметры сети (Network Options). Теперь, когда меню создано, вам нужно создать форму актуальных настроек.

```
// генерируем страницу настроек
function prowp_network_settings() {
    ?>
    <div class="wrap" >
        <div id="icon-options-general" class="icon32"></div>
        <h2>Network Settings</h2>
        <form method="post">
            <?php
            // загружаем параметры
            $network_settings = get_site_option( 'prowp_network_settings' );
            $api_key = $network_settings['api_key'];
            $holiday = $network_settings['holiday'];
            $rage_mode = ( ! empty( $network_settings['rage_mode'] ) )
                ? $network_settings['rage_mode'] : '';

            // создаем скрытое поле из соображений безопасности
            wp_nonce_field( 'save-network-settings', 'prowp-network-plugin' );
            ?>
            <table class="form-table">
                <tr valign="top"><th scope="row">API Key:</th>
                    <td><input type="text" name="network_settings[api_key]"
                        value="<?php echo esc_attr( $api_key ); ?>" /></td>
                </tr>
                <tr valign="top"><th scope="row">Network Holiday</th>
                    <td>
                        <select name="network_settings[holiday]">
                            <option value="halloween"
                                <?php selected( $holiday, 'halloween' ); ?> >
                                    Halloween
                            </option>
                            <option value="christmas"
                                <?php selected( $holiday, 'christmas' ); ?> >
                                    Christmas
                            </option>
                            <option value="april_fools"
                                <?php selected( $holiday, 'april_fools' ); ?> >
                                    April Fools
                            </option>
                        </select>
```

```
                </td>
            </tr>
            <tr valign="top"><th scope="row">Rage Mode:</th>
                <td><input type="checkbox" name="network_settings[rage_mode]"
                    <?php checked( $rage_mode, 'on' ); ?> /> Enabled
                </td>
            </tr>
        </table>
        <p class="submit">
            <input type="submit" class="button-primary"
                name="network_settings_save" value="Save Settings" />
        </p>
    </form>
</div>
<?php
}
```

Для управления сетевыми настройками вы будете использовать стандартную HTML-форму. Параметры будут сохранены как массив в единственном параметре, как описано в разделе «Настройки плагина» главы 8 «Разработка плагинов». Первый шаг — загрузка существующих значений настроек. Используя функцию `get_site_option()`, вы загрузите значение `prowp_network_settings`, являющееся массивом параметров, если таковые существуют. Затем поместите каждое значение параметра в индивидуальную переменную. Перед тем как вы действительно создадите форму, используйте функцию `wp_nonce_field()`, чтобы создать временное значение скрытого поля формы для безопасности.

Теперь пора строить форму. Первое поле формы — ключ API, стандартное текстовое поле. Второе поле формы — поле выбора. Заметьте, как вы используете функцию `selected()` для определения выбираемого параметра. Последняя настройка — флажок для Rage Mode. Этот параметр использует функцию `checked()`, чтобы определить, выбран ли параметр. Поскольку вы используете стандартную HTML-форму, нужно добавить кнопку отправки для отправки значений формы.

Теперь, когда вы установили форму сетевых настроек, нужно создать функцию для обработки и сохранения данных.

```
add_action( 'admin_init', 'prowp_save_network_settings' );
// сохраняем значения параметров
function prowp_save_network_settings() {

    // сохраняем параметры, если они были отправлены
    if ( isset( $_POST['network_settings'] ) ) {

        // проверяем из соображений безопасности
        check_admin_referer( 'save-network-settings', 'prowp-network-plugin' );

        // сохраняем параметры в переменной
        $network_settings = $_POST['network_settings'];

        // обрабатываем параметры
        $network_settings = array_map( 'sanitize_text_field', $network_settings );
```

```
        // сохраняем параметры
        update_site_option( 'prowp_network_settings', $network_settings );

    }

}
```

Для сохранения произвольной функцией данных формы будет использоваться зацепка `admin_init`. Первый шаг — проверка того, что значения формы отправлены. Если этого не произошло, обрабатывать нечего. Это можно сделать, проверив, что `$_POST['network_settings']` действительно задана, посредством функции PHP `isset()`. Как только вы убедились, что есть данные формы для обработки, нужно проверить временное значение с помощью функции `check_admin_referer()`.

После передачи временного значения данные будут храниться в переменной `$network_settings`. Поскольку обрабатываемые данные предоставлены пользователем, важно очистить их перед тем, как сохранять в базе данных. В этом примере вы будете использовать PHP-функцию `array_map()`, которая будет посылать каждое индивидуальное значение массива любой определенной функции, в данном случае `sanitize_text_field()`. Теперь, когда данные надлежащим образом очищены, можно сохранить параметр, используя функцию `update_site_option()`.

Готово! Вы только что построили полностью функциональный раздел сетевых настроек, который совместим со стандартным WordPress. Если пользователь использует Multisite, меню появится в Network Admin, а значения параметров будут храниться в таблице `wp_sitemeta`. Если пользователь не использует Multisite, меню появится в стандартной консоли администратора WordPress, а значения параметров будут храниться в таблице `wp_options`.

В листинге 10.3 приведен конечный вариант плагина.

Листинг 10.3. Сетевые настройки Multisite (prowp2-multisite-network-settings.zip)

```
<?php
/*
Plugin Name: ProWP2 Network Settings Example
Plugin URI: http://strangework.com/wordpress-plugins
Description: This is a plugin demonstrating the Multisite Network WordPress
     Settings
Version: 1.0
Author: Brad Williams
Author URI: http://strangework.com
License: GPLv2
*/
add_action( 'init', 'prowp_network_settings_menu' );
function prowp_network_settings_menu() {

    if ( is_multisite() ) {

        // Multisite включен, поэтому добавляем меню в Network Admin
        add_action( 'network_admin_menu', 'prowp_add_network_settings_menu' );
    } else {
```

```
        // Multisite не включен, поэтому добавляем меню
        // в панель администрирования WordPress
        add_action( 'admin_menu', 'prowp_add_network_settings_menu' );
    }
}
function prowp_add_network_settings_menu() {

    // добавляем меню настроек
    add_menu_page( 'Network Options Page',
        'Network Options', 'manage_options',
        'prowp-network-settings', 'prowp_network_settings' );
}
// генерируем страницу настроек
function prowp_network_settings() {
    ?>
    <div class="wrap" >
        <div id="icon-options-general" class="icon32"></div>
        <h2>Network Settings</h2>
        <form method="post">
            <?php
            // загружаем параметры
            $network_settings = get_site_option( 'prowp_network_settings' );
            $api_key = $network_settings['api_key'];
            $holiday = $network_settings['holiday'];
            $rage_mode = ( ! empty( $network_settings['rage_mode'] ) )
                ? $network_settings['rage_mode'] : '';

            // проверяем из соображений безопасности
            wp_nonce_field( 'save-network-settings', 'prowp-network-plugin' );
            ?>
            <table class="form-table">
                <tr valign="top"><th scope="row">API Key:</th>
                    <td><input type="text"
                            name="network_settings[api_key]"
                            value="<?php echo esc_attr( $api_key ); ?>"
                         /></td>
                </tr>
                <tr valign="top"><th scope="row">Network Holiday</th>
                    <td>
                        <select name="network_settings[holiday]">
                            <option value="halloween"
                                <?php selected( $holiday, 'halloween' ); ?>
                                 >Halloween</option>
                            <option value="christmas"
                                <?php selected( $holiday, 'christmas' ); ?>
                                 >Christmas</option>
                            <option value="april_fools"
                                <?php selected( $holiday, 'april_fools' ); ?>
                                 >April Fools</option>
                        </select>
                    </td>
                </tr>
                <tr valign="top"><th scope="row">Rage Mode:</th>
                    <td><input type="checkbox"
                         name="network_settings[rage_mode]"
```

```
                    <?php checked( $rage_mode, 'on' ); ?> />
                    Enabled
                </td>
            </tr>
        </table>
        <p class="submit">
            <input type="submit" class="button-primary"
                name="network_settings_save" value="Save Settings" />
        </p>
    </form>
</div>
<?php
}
add_action( 'admin_init', 'prowp_save_network_settings' );
// сохраняем параметры
function prowp_save_network_settings() {

    // сохраняем параметры, если они были отправлены
    if ( isset( $_POST['network_settings'] ) ) {

        // проверяем из соображений безопасности
        check_admin_referer( 'save-network-settings', 'prowp-network-plugin' );

        // сохраняем параметры в переменной
        $network_settings = $_POST['network_settings'];

        // обрабатываем параметры
        $network_settings = array_map( 'sanitize_text_field', $network_settings );

        // сохраняем параметры
        update_site_option( 'prowp_network_settings', $network_settings );

    }

}
```

Пользователи в сети

При работе с пользователями в сети Multisite всегда нужно проверять, что пользователь является членом определенного сайта. Чтобы сделать это, используйте функцию `is_user_member_of_blog()`.

```
<?php is_user_member_of_blog( $user_id, $blog_id ); ?>
```

Функция принимает два необязательных параметра. Первый — это идентификатор проверяемого пользователя. Если он не задан, функция проверяет текущего пользователя. Второй — идентификатор блога. Если он не задан, функция проверяет текущий сайт.

```
<?php
if ( is_user_member_of_blog() ) {
    // пользователь является участником этого сайта
}
?>
```

В приведенном выше примере кода проверяется, является ли пользователь членом просматриваемого сайта.

Теперь, когда вы понимаете, как проверить, что пользователь является членом сайта, можно добавлять пользователей на сайт с помощью функции `add_user_to_blog()`:

```
<?php add_user_to_blog( $blog_id, $user_id, $role ); ?>
```

Функция принимает три параметра:

1. `$blog_id` — ID сайта, на который вы хотите добавить пользователя;
2. `$user_id` — ID добавляемого пользователя;
3. `$role` — роль пользователя на сайте.

Теперь построим плагин, который автоматически добавляет авторизованного пользователя на любой сайт, посещаемый им в сети Multisite:

```
add_action( 'init', 'prowp_multisite_add_user_to_site' );
```

Сначала используем зацепку-действие `init`, чтобы выполнить произвольную функцию для добавления пользователя на сайт.

```
function prowp_multisite_add_user_to_site() {
    // перед работой проверяем, авторизован ли пользователь
    if( !is_user_logged_in() )
        return false;

    // загружаем ID текущего блога и данные пользователя
    global $current_user, $blog_id;
    // проверяем, не является ли пользователь участником этого сайта
    if( ! is_user_member_of_blog() ) {

        // добавляем пользователя на сайт в качестве подписчика
        add_user_to_blog( $blog_id, $current_user->ID, 'subscriber' );

    }

}
```

Первый шаг — убедиться, что пользователь авторизован, если это не так, выйти из функции возвращением `false`. Затем вызываются глобальные переменные `$current_user` и `$blog_id`. В них хранятся данные авторизованного в данный момент пользователя и Blog ID просматриваемого им сайта. Теперь нужно подтвердить, что пользователь не является членом этого сайта, используя функцию `is_user_member_of_blog()`. Последний шаг — добавление пользователя на сайт с помощью функции `add_user_to_blog()`. В данном примере вы задаете роль пользователя как подписчика, но ее можно легко заменить на любую другую.

Готово! Чтобы этот плагин работал по всей сети, нужно либо активировать его для сети, либо загрузить его в директорию `/mu-plugins`. Оба варианта запустят плагин для всех сайтов сети.

Законченный плагин показан в листинге 10.4.

Листинг 10.4. Автоматическое добавление пользователей на сайты в Multisite (prowp2-multisite-add-users.zip)

```
<?php
/*
Plugin Name: ProWP2 Multisite Auto-Add User to Site
Plugin URI: http://strangework.com/wordpress-plugins
Description: Plugin automatically adds the user to any site they visit
Version: 1.0
Author: Brad Williams
Author URI: http://strangework.com
License: GPLv2
*/
add_action( 'init', 'prowp_multisite_add_user_to_site' );
function prowp_multisite_add_user_to_site() {
    // перед работой проверяем, авторизован ли пользователь
    if( !is_user_logged_in() )
        return false;

    // загружаем ID текущего блога и данные пользователя
    global $current_user, $blog_id;
    // проверяем, что пользователь не является пользователем сайта
    if( ! is_user_member_of_blog() ) {

        // добавляем пользователя на сайт в качестве подписчика
        add_user_to_blog( $blog_id, $current_user->ID, 'subscriber' );

    }

}
?>
```

Теперь, кода вы понимаете, как добавлять пользователей на сайт, можно удалять их с сайта. Для удаления пользователей используется функция `remove_user_from_blog()`:

```
<?php remove_user_from_blog( $user_id, $blog_id, $reassign ); ?>
```

Она принимает три параметра:

1. `$user_id` — ID удаляемого пользователя;
2. `$blog_id` — ID блога, из которого удаляется пользователь;
3. `$reassign` — ID пользователя, которому переходят записи.

Параметры `$user_id` и `$blog_id` являются обязательными. Параметр `$reassign` — опциональный. Это должен быть ID пользователя, которому вы хотите перераспределить записи при удалении пользователя.

ЗАМЕЧАНИЕ

Помните, что добавление и удаление пользователя с сайта в Multisite не то же самое, что создание или удаление пользователей в WordPress, это просто добавление или удаление членства на сайте.

Чтобы вернуть список всех сайтов, к которым относится пользователь, используйте функцию `get_blogs_of_user()`. Она возвращает массив объектов, содержащий подробности по каждому сайту, доступ к которому имеется у пользователя. Например:

```
<?php
$user_id = 1;
$user_blogs = get_blogs_of_user( $user_id );
echo 'User '.$user_id.'\'s blogs:<ul>';
foreach ( $user_blogs AS $user_blog ) {
    echo '<li>' .$user_blog->blogname .'</li>';
}
echo '</ul>';
?>
```

Приведенный ранее код возвращает данные по всем сайтам, членом которых является пользователь с ID 1. Затем вы прогоняете возвращенный массив через цикл, отображая значение `blogname` для каждого сайта.

Суперадминистраторы

Ранее в этой главе речь шла о новой роли пользователя, появляющейся в Multisite, роли суперадминистратора. У любого пользователя-суперадминистратора есть полный контроль над каждым сайтом в сети Multisite, а также полный контроль над доступными плагинами и темами, всеми пользователями и настройками сети.

Чтобы вернуть список всех суперадминистраторов в Multisite, используется функция `get_super_admins()`. Она не принимает никаких параметров и возвращает массив имен пользователей всех суперадминистраторов в сети. Например:

```
<?php
$all_admins = get_super_admins();
print_r( $all_admins );
?>
```

Этот пример кода возвращает следующий массив суперадминистраторов:

```
Array
(
    [0] => admin
    [1] => michael_myers
)
```

Вы также можете проверить определенный идентификатор пользователя, чтобы определить, является ли пользователь суперадминистратором сети. Для этого используется функция `is_super_admin()`:

```
<?php
$user_id = 1;
if ( is_super_admin( $user_id ) ) {
    echo 'User is a Super Admin';
}
?>
```

Этот пример кода проверяет, является ли пользователь с ID 1 суперадминистратором. Функция принимает `$user_id` как необязательный параметр. Если ID пользователя не задан, функция проверяет текущего пользователя.

Теперь, когда вы понимаете, как проверять суперадминистраторов, вы можете сделать пользователя суперадминистратором. Эта роль передается существующему пользователю с помощью функции `grant_super_admin()`. Функция принимает один обязательный параметр: ID пользователя, назначаемого суперадминистратором.

```
<?php
$user_id = 34;
grant_super_admin( $user_id );
?>
```

Вы также можете легко лишить пользователя этой роли с помощью функции `revoke_super_admin()`. Как и в предыдущем коде, эта функция принимает один параметр: ID пользователя, лишаемого роли суперадминистратора:

```
<?php
$user_id = 34;
revoke_super_admin( $user_id );
?>
```

Обе эти функции находятся в `wp-admin/includes/ms.php`, то есть они недоступны публично и могут использоваться только администратором.

Сетевой статус

Multisite поддерживает различные функции для генерирования статуса сети. Функция `get_blog_count()` возвращает общее число сайтов в сети. Чтобы вернуть общее число пользователей, используется функция `get_user_count()`.

```
<?php
$site_count = get_blog_count();
$user_count = get_user_count();
echo '<p>Total sites: ' .$site_count .'</p>';
echo '<p>Total users: ' .$user_count .'</p>';
?>
```

Также можно применить функцию `get_sitestats()` для возвращения обоих значений одним массивом.

```
<?php
$network_stats = get_sitestats();
echo '<p>Total sites: ' .$network_stats['blogs'] .'</p>';
echo '<p>Total users: ' .$network_stats['users'] .'</p>';
?>
```

Схема базы данных Multisite

WordPress Multisite поддерживает различные схемы баз данных из стандартного WordPress. При активации Multisite WordPress создает необходимые таблицы в базе данных для поддержания функциональности Multisite.

Специфические таблицы Multisite

WordPress хранит глобальные настройки Multisite в централизованных таблицах. Они создаются, только когда Multisite активирован и установлен, за исключением таблиц `wp_users` и `wp_usermeta`, существующих в стандартном WordPress.

- `wp_blogs` — содержит каждый сайт, созданный в Multisite.
- `wp_blog_versions` — содержит текущую версию базы данных каждого сайта в сети.
- `wp_registration_log` — журнал всех пользователей, зарегистрированных и активированных в WordPress.
- `wp_signups` — содержит пользователей и сайты, зарегистрированные через процесс регистрации WordPress.
- `wp_site` — содержит информацию по адресу первичного сайта.
- `wp_sitecategories` — содержит глобальные элементы. Существует, только если они были активированы в WordPress.
- `wp_sitemeta` — содержит данные параметров сети, включая учетные записи суперадминистраторов.
- `wp_users` — содержит всех пользователей, зарегистрированных в WordPress.
- `wp_usermeta` — содержит все метаданные для учетных записей пользователей в WordPress.

Как вы уже, возможно, заметили, некоторые важные таблицы WordPress отсутствуют. Оставшиеся таблицы, созданные для Multisite, специфичны для сайтов.

Специфические таблицы сайтов

У каждого сайта в сети есть собственные специфические таблицы баз данных. Они содержат контент и настройки для данного сайта. Помните, что префиксом таких таблиц является значение `$table_prefix`, определенное в `wp-config.php`, за которым идет `$blog_id` и имя таблицы.

- `wp_2_commentmeta`;
- `wp_2_comments`;
- `wp_2_links`;
- `wp_2_options`;
- `wp_2_postmeta`;
- `wp_2_posts`;
- `wp_2_terms`;

- `wp_2_term_relationships`;
- `wp_2_term_taxonomy`.

Каждый раз, когда вы создаете новый сайт в сети Multisite, WordPress строит в базе данных указанные ранее девять таблиц для нового сайта. Как видите, они могут быстро увеличить базу данных в размере. Вот почему единственным ограничением WordPress Multisite являются ресурсы сервера, доступные для вашей сети сайтов. Если в сети 1000 сайтов, в базе данных будет на 9000 таблиц больше. Очевидно, что сеть такого размера не будет работать нормально на маленьком хостинге общего пользования.

В главе 6 «Управление данными» мы говорили о важности использования класса базы данных WordPress при прямом запросе базы данных. Это особенно важно в Multisite, поскольку префиксы таблицы содержат Blog ID просматриваемых сайтов. При написании произвольного запроса ссылку на таблицу всегда нужно предварять `$wpdb->`, который добавит ID сайта, если вы используете Multisite. Например, `$wpdb->posts` запросит таблицу `wp_2_posts`, убедившись, что вы работаете на Blog ID 2 в сети.

Резюме

WordPress Multisite — удивительная возможность WordPress с безграничными возможностями. Теперь, когда Multisite — возможность ядра WordPress, многие пользователи превращают стандартный сайт на WordPress в сеть Multisite, чтобы пользоваться преимуществами быстрого построения сайтов и возможностями сети. Поскольку все больше и больше пользователей знакомятся с Multisite, широта его использования также быстро увеличивается. При разработке для WordPress важно думать о Multisite и о том, как ваш код может применяться для его мощных возможностей, рассмотренных в этой главе. А пользователю WordPress важно проверять, что используемые темы и плагины совместимы с Multisite.

В следующей главе речь пойдет об агрегировании контента. Вы научитесь работать с внешними API для импорта данных из различных источников на веб-сайт, интегрировать кнопки социальных медиа и понимать различные рекламные методы, а также возможности монетизации сайта.

11 Агрегация контента

В этой главе:

- Как сделать ваш контент заметным
- Импорт различных источников на ваш сайт
- Использование WordPress для кэширования контента из удаленных источников
- Различные методы рекламы, полезные для монетизации вашего сайта

Когда мы начинали писать эту книгу, «агрегация контента» рассматривалась как набор механизмов, необходимых для обновления вашего сайта на WordPress за счет информации, получаемой из постоянно появляющихся социальных сетей, а также для экспорта обновленного контента вашего сайта в эти сети. Иногда WordPress служил отправной точкой, а иногда — пунктом назначения, но внимание всегда было сосредоточено на круговороте контента, причем некоторые аспекты этой темы были рассмотрены в главе 1, а именно: зачастую контекст не менее важен, чем само содержимое. «Агрегация контента» — это то, как и куда вы хотите направить внимание ваших посетителей и с какой целью.

При кратком рассмотрении того, что изменилось за время, прошедшее между разными изданиями этой книги, мы обнаруживаем следующее: Facebook набрал практически миллиард пользователей; Twitter вывел в тренд использование хэштегов, а большинство крупных сайтов обзавелись пафосными сокращенными адресами, вытесняющими `bit.ly`. Google, Bing и Yahoo! можно считать хоть и не равноправными, но тремя столпами интернет-поиска.

И только одна вещь остается неизменной: ваш сайт на WordPress представляет пользователям квинтэссенцию вашего опыта, собственной торговой марки, рекомендуемого контента и дизайна. Именно его адрес вы при регистрации вносите в поле Личный сайт (Personal Website) или Рабочий сайт (Business Website), и именно его особенности позволят вам определить ваш личный способ агрегации

контента. Если вы делаете сайт для себя и о себе и видите WordPress лишь как дополнительный инструмент общения с посетителями, то вы будете вытягивать содержимое из других ресурсов с целью популяризации вашего сайта. Вашей целью обычно является максимальное привлечение публики вне зависимости от того, где именно эта публика «обитает» в Сети. С другой стороны, если ваш сайт посвящен вашей работе или вашей собственной торговой марке консультанта или эксперта или же он является просто «витриной» для ваших драгоценных рекламодателей, вы, несомненно, желаете сосредоточить внимание публики именно на самом сайте, а Facebook и Twitter служат лишь дополнительными посредниками, привлекающими читателей, которых необходимо заставить просматривать ваш сайт на WordPress.

Когда вы будете с помощью данной главы оценивать методы агрегации входящей и исходящей информации, подумайте о том, как они повлияют на вашу возможность «дотянуться» до публики. Является ли WordPress единственным каналом или всего лишь одним из многих?

В данной главе обсуждается перемещение контента из внешних источников, обычно социальных сетей, на ваш сайт и, соответственно, экспорт вашего контента и публикация его через сторонние сайты и ленты. В особенности вам стоит подумать о том, как охватить YouTube, Twitter, Facebook и ленты для подписчиков, так как они являются наиболее распространенными. Вам также стоит обдумать использование других сайтов с полезным контентом, особенно имеющих открытые API, такие как Google Maps, и то, как вы сможете применить API WordPress для интеграции подобных источников. Вам стоит начать с использования социальных сетей и настройки исходящего контента, чтобы привлечь внимание к своему сайту.

Привлечение внимания

Практически все сайты работают так, что их посетители вовлекаются в бизнес: будь то продажа товаров, услуг или вашей собственной торговой марки. Однако до того, как это вовлечение произойдет, посетителям необходимо получить возможность найти ваш сайт. Данный раздел посвящен продвижению вашей торговой марки и вашего сайта в Сети посредством совместного использования контента и социальных сетей, а не с помощью оптимизации сайта для поисковых машин, известной как SEO (Search Engine Optimization — поисковая оптимизация). SEO и то, как можно упростить обнаружение вашего сайта, описаны в главе 12.

Продвижение вашего сетевого образа является одним из главных поводов сосредоточить все свои онлайн-взаимодействия в одном месте. Вы можете сконцентрировать свое взаимодействие с социальными носителями информации на вашем сайте, для того чтобы представить свою участие в сообществе или работу в наилучшем свете. Это также позволит продемонстрировать ваш профессионализм в одной или нескольких специфических областях и расширить круг ваших потенциальных посетителей. Это действительно служит методом установления деловых контактов с различными группами читателей.

Даже если вы не используете свой публичный образ для работы, централизация вашей сетевой активности сыграет на руку вашим увлечениям и интересам. Если вы состоите в социальных сетях в группах, посвященных варению пива в домашних условиях, то почему бы не собирать всю информацию об этом в одном месте? Если же вы привлекаете внимание своими остроумными догадками или умеете делиться точной и полезной информацией, то агрегация этих сведений — путь к тому, чтобы стать признанным экспертом в интересующей вас области. Приятным последствием агрегации является следующее: чем больше ссылок указывают на ваш сайт, тем более популярным считают его поисковые машины; это описано в главе 12.

Сбор информации с различных сайтов и размещение ее на вашем сайте облегчает для других поиск этой информации. Ваши читатели или потенциальные посетители не должны держать кучу открытых вкладок для каждого из тех мест, где транслируются или используются ваши обновления. Кроме того, как ваши читатели посмотрят новое рекламное видео на нашем канале YouTube, если они даже не могут узнать о том, что оно вышло? Хранение всей информации на ее первоисточнике позволяет предоставить все описанные различные типы данных вниманию ваших пользователей с помощью агрегации контента. И в конце концов это привлекает трафик на ваш сайт, так как он становится единственным истинным источником информации.

Это классическая проблема успешности контента, и она заслуживает некоторого обсуждения. Ваш сайт — лишь один из сотен миллионов существующих источников содержимого. Однако люди, с которыми вы встречаетесь и активно общаетесь, или группа «друзей и их родственников» (или «замыкание множества», если вдруг мы говорим сейчас с математическими маньяками) все же представляют собой некую аудиторию, *вашу* личную аудиторию. Агрегация контента позволяет расширить вашу аудиторию за счет того, что она *возникает* в разных местах с подходящим контентом, контекстом и подробными обновлениями. Создание твитов о новых записях в блоге или импорт цитат из записей Facebook, например, являются простыми способами распространения информации. Дальнейшей целью будет привлечение профессиональных организаций, которые будут работать на укрепление вашей репутации.

Кроме того, агрегация — это не только аккумуляция контента социальных сетей, это и использование сайтов, обладающих определенным функционалом, геофизическим контентом или службами, за счет встраивания на ваш сайт. Google Maps (Карты Google) — распространенное дополнение, позволяющее заказчикам и клиентам найти ваш офис; таким же полезным дополнением является включение в Twitter обсуждений, посвященных вашей торговой марке или содержащих соответствующий хэштег.

В конце концов, расширение функционала вашего сайта — еще один маркетинговый ход. Вам стоит либо пытаться сделать свой сайт многофункциональным и привлекательным для потенциальных клиентов и заказчиков, либо же интегрировать в него выгодный внешний источник контента.

Кнопки социальных сетей

Маркетинг есть маркетинг, и вам нужно как-то сделать свой сайт заметным. Хорошим (и, возможно, лучшим) способом добиться того, чтобы ваш контент замечали, а приток трафика к вашему сайту увеличился, является использование всей мощи сайтов социальных сетей. Во-первых, вашему сайту необходим качественный интересный контент. Однако, в отличие от истории, рассказанной в фильме *Поле чудес (Field of Dreams)*, если даже вы все сделаете, то не факт, что все получится. Вам нужно рекламировать свой сайт. Само собой, амбициозный и преданный посетитель может взять ссылку на ваш сайт и распространить ее по Сети, но почему бы не облегчить ему задачу?

Добавление кнопок социальных сетей на ваш сайт на WordPress облегчит задачу внесения вашего контента в свои перечни, рейтинги и массивы или же упростит создание банальных сообщений для публики: «Эй, ребята, мне это понравилось!». Стимулирование посетителей делиться своими предпочтениями с обратной отсылкой к вашему контенту — ключевой аспект успешности этого контента; без рекомендаций от одинаково настроенных людей никто и никогда не найдет ваше содержимое. Этот принцип работает для музыкальных произведений, фильмов и блогов. Так почему бы вам не создать функцию, позволяющую одним щелчком мыши рассказать о вашем контенте в лентах посетителей сайта?

Плагин ShareThis (http://wordpress.org/extend/plugins/share-this/) делает именно это. Данный плагин поддерживает ссылки примерно на 120 каналов социальных сетей, через которые ваши посетители смогут поделиться контентом. Этот плагин легко настраивается и имеет удобную панель управления. Вы можете указать, через какие сайты можно делиться информацией, что позволит вам создать список сайтов, являющихся целевыми для ваших посетителей в соответствии с вашими личными предпочтениями. Есть также несколько функций, позволяющих контролировать отображение иконок социальных сетей на вашем сайте, присутствуют и некоторые другие возможности настройки стиля.

По умолчанию включена весьма удобная строка с краткой информацией в «подвале» каждой записи, как показано на рис. 11.1.

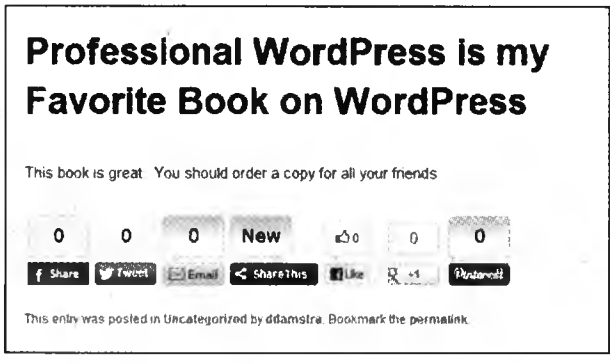


Рис. 11.1. Кнопки социальных сетей под записью WordPress

Одним из главных достоинств описанного плагина является его простота. Он начинает работать сразу после установки и работает именно так, как сказано в его описании. Посетители могут прочесть ваш чудесный контент и поделиться им на Pinterest. Для этого им просто нужно щелкнуть мышью по иконке и войти в учетную запись Pinterest. Вот так все просто!

Делимся контентом

Если вы решили, что ваш сайт на WordPress является главным пунктом назначения для ваших читателей, то их переход на ваш сайт из соцсетей Facebook, Twitter и Pinterest можно условно назвать движением против течения. Пока посетители делятся вашим контентом с помощью кнопок социальных сетей, вы также можете отправлять целые записи, их лиды или заголовки в профили ваших других сетевых учетных записей.

Информирование о ваших обновлениях через Twitter может быть по вашему желанию как простым, так и сложным. В крайнем случае вы можете просто писать твиты о ваших новых записях, используя сокращенные сетевые адреса, что позволит создать в Twitter пространство хэштегов, комментариев или хитрых перекрестных ссылок. Если же вы хотите, чтобы ваша учетная запись в Twitter сообщала о каждой новой записи, появляющейся на сайте WordPress, воспользуйтесь плагином, подобным Tweetily (http://wordpress.org/extend/plugins/tweetily-tweet-wordpress-posts-automatically/), позволяющим создавать твиты с заголовками записей и их лидами или же с хэштегами.

Если вы отправляете свой контент в Facebook, возникает вопрос: куда я хотел бы направить своих читателей? Если вы автоматически копируете контент сайта на WordPress в вашу новостную ленту Facebook, то у читателей Facebook отпадает необходимость возвращаться на ваш сайт, что может привести к потере комментариев, посетителей или возможных просмотров вашего содержимого. Вы рискуете существенно сократить круг ваших читателей, делясь содержимым, как бы противоречиво это ни звучало. С другой стороны, если ваши читатели в Facebook служат источником новых посетителей, которые могут узнать о вас от друга или от друга друга либо просто из-за того, что кто-то нажал «Я рекомендую» под вашей записью, то отправка контента в Facebook может служить инструментом расширения круга ваших читателей. Нужно найти золотую середину: если вы все же решите создать страницу Facebook для вашего сайта на WordPress, стоит тщательно рассмотреть двойственную природу понятия «друзья» и ограничить количество друзей, которых вы можете завести.

Даже если страничка является вашей личной учетной записью, самым простым способом передать контент с сайта на WordPress в Facebook будет приложение Facebook RSS Graffiti (`http://apps.facebook.com/rssgraffiti/`). Находясь в Facebook, установите приложение, а затем направьте его на RSS-ленту вашего сайта в форме `example.com/feed/rss2`. В приложении RSS Graffiti легко настроить частоту проверки RSS-ленты на предмет обновлений, а также заголовки новостей,

предшествующие каждому обновлению. Помимо этого присутствует очень важная возможность настроить импорт записи целиком или же только ее отрывка. Если вы посылаете выдержки из WordPress в Facebook, то пользователи Facebook, заинтересованные вашим контентом, будут вынуждены просматривать ваш сайт для получения доступа к полному объему содержимого. Если вы боитесь потерять читателей, пользуйтесь отрывками, если же вы пытаетесь увеличить свое присутствие в социальных сетях, отправляйте в обратный поток полноценные записи.

Кнопки, значки или и то и другое?

Как вы видите, у совместного использования контента есть две равноценные цели. Первая — распространение вашего контента через социальные сети, такие как Facebook, Twitter и Pinterest, которые уже рассматривались в данном разделе. Вторая — привязка ссылок с вашего сайта к вашему личному профилю в социальных сетях, четко дающая понять, что стоит за словами «Следите за нами через Twitter» (Follow me on Twitter) или «Найдите нас в Facebook» (Find us on Facebook) на кнопках, которые вы, возможно, разместили с помощью виджета WordPress в одной из боковых панелей вашего сайта.

Ссылка на сайте на ваши профили в социальных сетях способствует реализации идеи использовать ваш сайт в качестве центральной точки. Это также подтверждает то, что заявленные профили действительно являются вашими и служат для представления вашего мнения в Сети. Связывание этих профилей в единое целое — палка о двух концах, так как оно не только служит для подтверждения ваших профилей и укрепления репутации в Сети, но и направляет читателей с одного сайта к вашему профилю или пункту назначения на другой, что потенциально сокращает вашу аудиторию. Направление, в котором движется трафик, — это выбор, который вам предстоит сделать, и с ним тесно связаны опасения, высказанные в начале данной главы.

Валидация ваших профилей в сообществах — прекрасная вещь, если вы используете свое присутствие в Сети для успешной работы как отдельного человека, так и целой организации. Она позволяет рекламировать ваше участие в работе и ее масштабы. Подтверждение своей личности через основной сайт позволяет проверить и прочие сетевые учетные записи. Это также избавляет посетителя вашего сайта от любых возможных сомнений, а вас — от работы со вкладкой Проверка учетных записей (Verified Accounts) Twitter, а еще вам не придется менять имя на @theRealDavidDamstraHonest.

Вы можете очень легко создавать ссылки на свои профили, если отформатируете шаблоны файлов или воспользуетесь HTML-виджетами. В целом эти ссылки меняются очень редко, почти никогда. Тем не менее существует несколько удобных плагинов, которые выполнят эту тяжелую работу за вас.

Например, плагин Social Media Widget (http://wordpress.org/extend/plugins/social-media-widget/) Брайана Фрейтага, который поддерживает многие социальные сети. После установки плагина перетащите виджет в подходящую боковую панель панели

управления. Настройте виджет на адреса тех социальных сетей, в которых вы публикуете свой контент. На рис. 11.2 приведен пример этого виджета.

Рис. 11.2. Виджет для боковой панели Social Media Widget

Помимо встроенных сайтов социальных сетей данный плагин позволяет использовать до шести дополнительных сетей по вашему выбору, а также разрешает добавлять ваши собственные иконки. Это отличный способ сосредоточить всю свою сетевую активность на одной легкоуправляемой странице.

Есть еще несколько плагинов, позволяющих добиться аналогичных результатов. Попробуйте их и сделайте собственный выбор.

Простые значки социальных сетей

Существует два различных способа интеграции внешних сайтов социальных сетей в ваш сайт, и каждый из этих способов по-своему влияет на ваш контент. Вам нужно научиться отличать эти способы и принять решение, в каком случае какой из них вам необходим.

Первый называется «простые значки социальных сетей». Они представляют собой моментальные снимки вашей активности на стороннем сайте, сделанные в конкретный момент времени, и могут автоматически генерироваться целевым сайтом (обычно это кнопка, предлагающая добавить значок). Контент, скрывающийся за значками, меняется, как только вы совершаете некое действие на стороннем сайте. Например, эти значки могут представлять собой виджеты для боковой панели, показывающие ваши последние записи в Twitter или статус в Facebook.

Эти простенькие значки социальных сетей демонстрируют ваше участие в работе других сайтов, но не вносят никакого вклада в контент вашего. Они играют скорее декоративную, чем содержательную, роль, даже если вы весьма активны на других сайтах. Как было сказано ранее, если, например, в Twitter вы более активны, чем в создании обновлений для своего блога, использование соответствующего значка может привести к миграции некоторых читателей, желающих наблюдать за вашей сетевой деятельностью, на вашу страницу в Twitter.

Вторым вариантом является извлечение контента или его фрагментов из некоего места и публикация его на вашем сайте на WordPress. Это дает контенту собственную жизнь, но на ваших условиях. Особенно важный момент: вы не полагаетесь на то, что третья сторона сохранит ваш контент, а создаете собственную копию

в собственной базе данных, за поддержание которой вы же и несете ответственность. Это часто называют «владеть данными».

И моментальные снимки, и публикация сторонней информации имеют определенную ценность, но вы должны осознать различия в стабильности и долговечности результатов использования каждого из этих методов. В данной главе рассматриваются оба подхода.

Использование значков социальных сетей включает ваш сайт WordPress в цикл положительной обратной связи. В идеале, если существуют люди, следящие за вами через Twitter или Facebook, то велика вероятность того, что они будут просматривать ваш сайт с целью прочитать контент в большем и более полном объеме. Люди, читающие ваш сайт, могут также захотеть следить за вашими более краткими и отвлеченными обновлениями на других сайтах; в обоих случаях вы надеетесь, что друзья ваших друзей привлекут внимание к контенту.

Сбор внешнего контента

Вы уже оценили достоинства и недостатки интеграции стороннего контента в ваш сайт на WordPress. Следующий вопрос, на который стоит ответить: как вы собираетесь это сделать? Какие сайты вы включите в список?

В принципе, вы можете добавить только те сайты. которые имеют API для сбора данных. Это не совсем верно, так как вы можете сами написать поисковый бот, который будет заходить на удаленный сайт и собирать необходимую вам информацию, но это довольно сложно и накладно с точки зрения поддержки такого механизма. Поэтому в данной книге мы собираемся сосредоточить внимание на сайтах с открытыми API.

Один из способов интегрировать службы — это прочитать документацию по API и создать плагин или другую функцию, позволяющую извлекать и конвертировать информацию в ту форму, которую может использовать WordPress, например запись. Так как внутренняя архитектура WordPress представляет собой PHP, вы можете использовать любые PHP-трюки, которые есть в вашем распоряжении для написания подходящего кода. Более подробную информацию о разработке плагинов вы можете получить из главы 8. Честно говоря, вам даже не нужно использовать PHP. Вы можете использовать любой промежуточный язык или интерфейс для работы непосредственно в таблице WordPress.

В данном разделе мы сосредоточимся на разработке рабочего сайта, который будет включать социальные сети и другие сайты, имеющие функции, которые могут быть встроены в ваш сайт и положительно повлияют на его ценность или контекст. Вы можете разработать некоторые функции самостоятельно, однако гораздо проще воспользоваться готовыми решениями, особенно если они бесплатны.

Начиная с версии 2.9 в WordPress включена поддержка формата oEmbed. oEmbed — это инструмент, позволяющий одному сайту интегрировать фрагмент кода с другого сайта. Термин «интегрировать» используется здесь в достаточно широком смысле,

так как HTML может представлять собой просто элемент `<iframe>`, однако использование этой функции уменьшает усилия, необходимые, например, для включения видео с YouTube в вашу запись WordPress. Обратитесь к кодексу WordPress для получения списка служб, которые полностью поддерживаются oEmbed.

Интеграция видео с YouTube

Предположим, у вас есть замечательное новое рекламное видео, которое вам необходимо добавить и растиражировать посредством своего сайта на WordPress. До того как вышла версия WordPress 2.9, вам необходимо было обращаться на сайт YouTube и копировать код для встраивания, предоставляемый YouTube, после чего вы должны были вставить его в окошко просмотра кода на панели **Write (Написать)** вашей записи или страницы WordPress. Если честно, это было не так уж ужасно — обычный «копипаст», известный большинству из нас. Тем не менее среднестатистические администраторы сайтов, которые были не так близко знакомы с хитростями HTML, часто испытывали трудности.

Так как YouTube поддерживает протокол oEmbed, вы можете использовать это свойство WordPress, чтобы избежать процесса копирования HTML-кода и просто вставить URL-видео в вашу запись. Этот URL должен находиться на отдельной строке и не должен быть гиперссылкой, то есть вам, возможно, придется снять ссылку в визуальном редакторе в основном поле для того, чтобы она заработала (рис. 11.3).

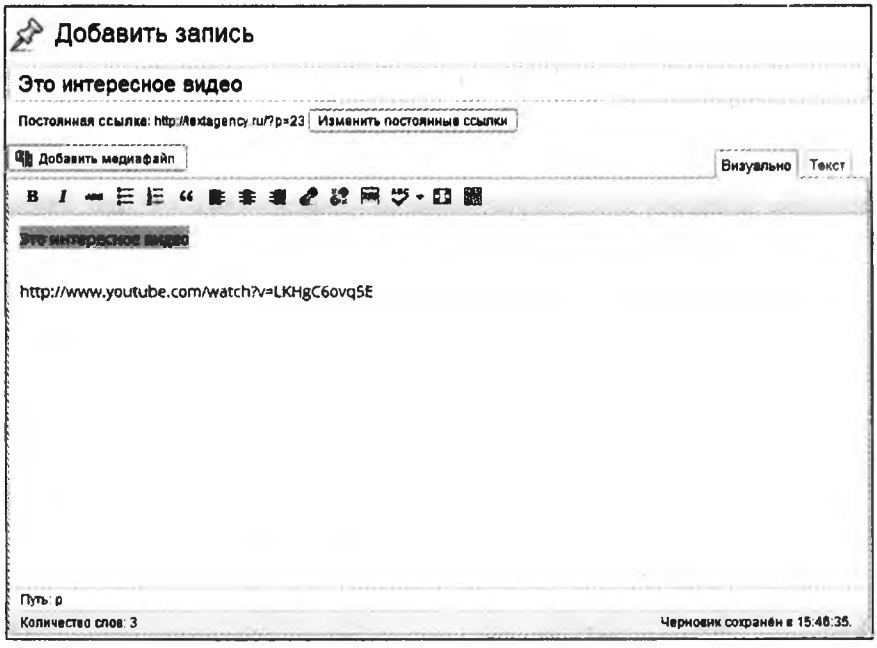


Рис. 11.3. Адрес с YouTube в основном поле записи WordPress (обратите внимание на отсутствие гиперссылки)

При отображении вашей записи через браузер, WordPress опознает электронный адрес как встроенный через oEmbed и обратится к контенту с YouTube для замещения адреса реальным видео, как показано на рис. 11.4.

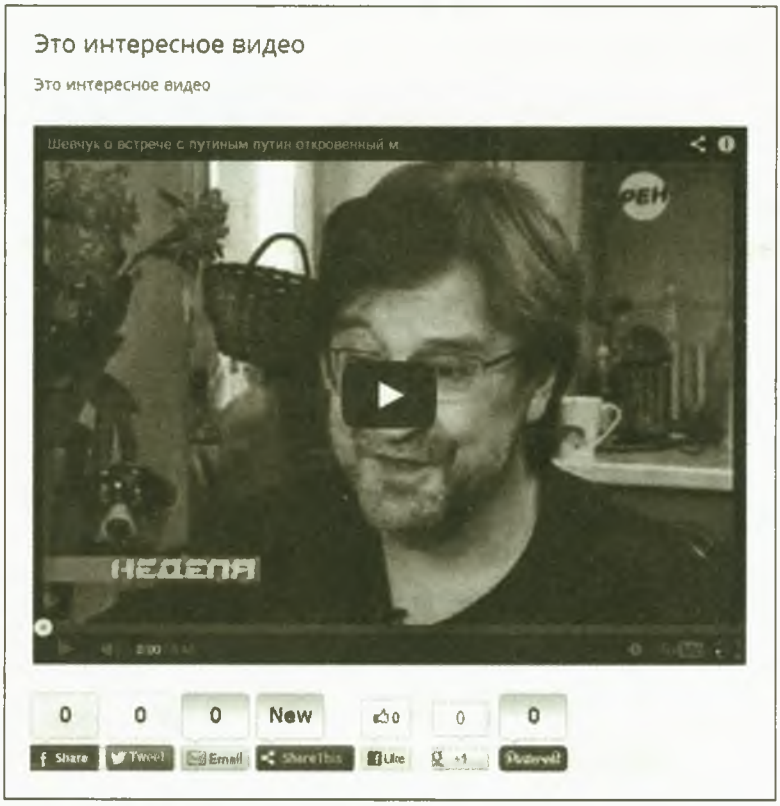


Рис. 11.4. Запись WordPress со встроенным видео с YouTube

Трансформация происходит как по волшебству. Если вы посмотрите код HTML, то заметите, что WordPress использует HTML-элемент `<iframe>` для проигрывания видео с YouTube. Это просто идеально, так как при этом реальный видеоконтент поступает в браузер пользователя прямо с серверов YouTube, которые оптимизированы для потоковых данных.

Интеграция Twitter

oEmbed начал поддерживать Twitter не так давно — в версии WordPress 3.4. Как и в случае с YouTube, вы можете добавить диалог из Twitter в запись или страницу WordPress, просто прописав его URL отдельной строкой и позволив WordPress и oEmbed сотворить чудо. Например, запись WordPress, приведенная на рис. 11.5, отобразится в браузере в виде, показанном на рис. 11.6.

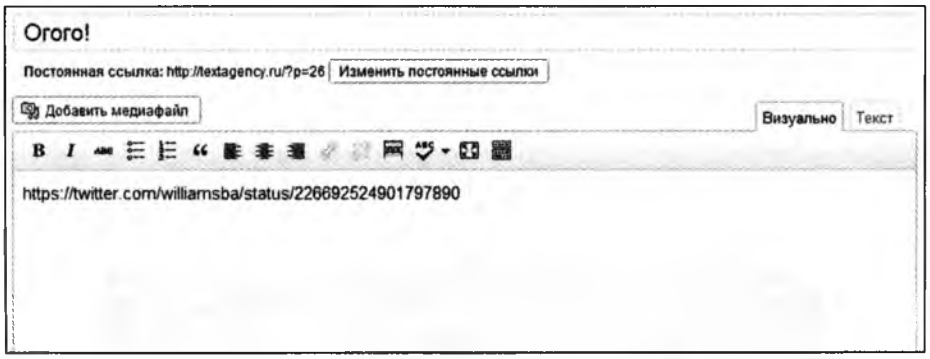


Рис. 11.5. Панель записей WordPress с электронным адресом Twitter (обратите внимание на отсутствие гиперссылки)

Рис. 11.6. Отображение диалога из Twitter в браузере

Функционал oEmbed практически безграничен. Если вам нужно перенести отдельную запись из Twitter или моментальный снимок с сайта третьей стороны, то возможно и это. Однако Twitter являлся примером для всех открытых API. Остаются ли они до сих пор открытыми? По API Twitter существует достаточно документации, и его легко использовать сразу после авторизации. Кроме того, у него есть огромное количество возможностей. Всё это превращает интеграцию Twitter в ваш сайт на WordPress в легкую задачу. Существует несколько вещей, которые вы можете сделать с интеграцией Twitter. Например, вы можете выводить ваши последние твиты в виджете боковой панели. Это пример простейшего взаимодействия с социальными сетями. Вы можете архивировать каждую запись в Twitter в отдельной записи WordPress или пойти дальше и создавать еженедельные или ежедневные архивы. Вы можете брать отдельные записи в Twitter и использовать их в качестве заголовков для создания эффекта динамичности или

же, в конце концов, вы можете настроить интеграцию в обратном направлении и автоматически создавать новую запись в Twitter каждый раз, когда вы публикуете запись в своем блоге.

Плагин Алекса Кинга — Twitter Tools (http://wordpress.org/extend/plugins/twitter-tools/) — выполняет большинство из описанных функций. Все они содержатся в одном плагине с простой панелью управления, и вы можете выбрать, как его использовать. Если честно, данный плагин достаточно сложен в установке, но беда не в плагине, а в том, как Twitter изменил доступ к API. Во время выхода первого издания этой книги API Twitter был полностью открытым, и интеграция казалась очень легкой. Вы могли задействовать API Twitter множеством различных способов, так как он был простым и мощным. Теперь Twitter блокирует некоторые вещи, делая интеграцию все более затруднительной. Однако, после того как вам удастся пробраться через окна настроек безопасности, вы получите доступ ко всему прекрасному функционалу этого плагина.

Например, такие функции, как отображение ваших последних записей в Twitter в простом виджете социальных сетей для боковой панели, легкодоступны при использовании этого плагина. Сначала установите плагин на ваш сайт на WordPress и активируйте его. С помощью новой панели инструментов Twitter Tools разберитесь со всеми настройками безопасности Twitter и авторизуйте ваше новое соединение. Twitter называет его app (приложение), и если следовать указаниям, то всё будет работать. Далее, сохраните настройки и перейдите в панель управления виджетами. Вы увидите новый виджет, который по умолчанию выводит три последние записи Twitter. Перетащите виджет на нужное место в боковой панели вашей темы для того, чтобы его активировать. Преимуществом использования такого метода является простота применения: включите его, и он работает. Недостатком же является кратковременная природа такого типа интеграции. Вы можете вызвать у людей интерес и заставить их наблюдать за вами в Twitter, но это не создаст долгосрочного контента на вашем личном сайте.

Twitter Tools позволяет конвертировать каждую отдельную запись в Twitter в запись вашего блога. В зависимости от того, как вы используете Twitter, это может оказаться полезным для создания резервных копий или архивов ваших твитов на вашем собственном сайте. Например, если вы применяете Twitter для создания заметок для себя, возможность WordPress конвертировать их в записи блога позволит вам совместить простоту Twitter и всю мощь структурированного контента WordPress.

Альтернативным подходом является публикация ваших твитов с помощью типа записи «заметка» (aside). К сожалению, на данный момент это невозможно, так как плагин не поддерживает его и будет смешивать твиты с обычными новостными записями. Думайте о заметках так же, как о комментариях, которые вы добавляете в диалог и которые не относятся к основной обсуждаемой теме. В таком случае и по причине того, что они исходят из Twitter, они будут представлять собой записи из одной строки, которые идеально подходят для формата заметок, если бы он был доступен. Просто возьмите записи из Twitter и создайте новые записи

в блоге в отдельной рубрике. Ваша тема должна иметь возможность работать со специфической логикой для создания правильного стиля заметок, отображаемых в боковой панели или подсвечиваемых для того, чтобы отделить их от основных записей в блоге. После того как вы превратили записи в Twitter в новые записи блога, вам остается просто использовать тему Twenty Eleven, обсуждаемую в главе 9, и немного специально разработанного CSS для того, чтобы перевести их в формат заметок.

Если вы публикуете каждую запись в Twitter в качестве индивидуальной записи в блоге, Twitter Tools дает возможность создавать еженедельные или ежедневные дайджесты. Это особенно интересно в том случае, если вы используете твиты для описания каких-то моментов вашей ежедневной жизни. Сложность использования Twitter состоит в том, что, с одной стороны, это очень просто, а с другой — по причине ограничения размера записей — вы вынуждены делать их очень короткими. Это и есть микроблогинг. В этом случае вам приходится рассказывать про ваш день с помощью серии маленьких записей. Возможность WordPress импортировать их создает другую форму для того же повествования. Это ничем не отличается от создания и публикации полновесной записи блога, за исключением того, что при использовании этого подхода записи формата микроблогов, используемого Twitter, автоматически собираются в цельную запись блога.

Другая возможность Twitter Tools позволяет вам направлять интеграцию в обратную сторону, то есть создавать запись в Twitter каждый раз, когда вы публикуете новую запись в блоге. Эта возможность должна быть полностью включена через панель инструментов Twitter Tools. После включения этой возможности плагин Twitter Tools будет автоматически создавать от вашего имени запись о том, что новая информация была опубликована в вашем блоге, и присоединять ссылку на вашу статью. Для некоторых людей это является достойной альтернативой лент RSS, которые мы рассмотрим далее в этой главе.

Google Maps

Google Maps (Карты Google), как, в принципе, любая картографическая служба, — достаточно востребованная вещь. Если по какой-то причине вы хотите, чтобы ваши заказчики лично посетили место вашей работы или некое событие, вам нужно объяснить им, как добраться. Сетевые картографические службы и службы создания маршрутов, такие как Google Maps, сейчас широко распространены. Вы уже не сможете вспомнить, как вы находили дорогу в городе до того, как они появились, и в то же время вы можете получать разнообразную пользу от служб установления местоположения.

В любом случае, вам нужно добавить на ваш сайт карту. Google предоставляет отличный механизм для простого встраивания карты в ваш сайт с помощью своих инструментов. Вы можете найти эту функцию в правом верхнем углу страницы Google Maps. Скопируйте код и вставьте его в вашу запись или на вашу страницу WordPress. Это просто и это работает.

Google Maps кажутся одним из первых кандидатов на поддержку oEmbed, однако они не поддерживаются. Так что на данный момент вам придется довольствоваться старым добрым и надежным копированием-вставкой для встраивания отдельных карт на ваш сайт на WordPress.

Как и Twitter, Google Maps имеют открытые API, и существует множество плагинов WordPress, которые включают возможности или функционал для работы с картами, такие как размещение нескольких флажков на одной и той же карте, встраивание нескольких карт в одну запись или пользовательский HTML-интерфейс для администратора вашего сайта. В зависимости от ваших потребностей или потребностей ваших клиентов вы можете воспользоваться любой предоставленной возможностью.

Интеграция Facebook

Facebook — это новая закрытая система среди сетевых сообществ. Много путей ведут на Facebook, но очень мало существует возможностей извлечь оттуда данные. Это делает интеграцию сайта на WordPress с Facebook довольно сложной задачей. До последнего времени Automattic и Facebook работали вместе над созданием плагина специально для интеграции WordPress и Facebook; он доступен в Сети по адресу http://wordpress.org/extend/plugins/facebook/.

Плагин нацелен на глубокую интеграцию сети Facebook с вашим сайтом на WordPress. Как и в случае с Twitter, вам необходимо зарегистрировать Facebook App на Facebook для того, чтобы использовать этот плагин. Заставить интеграцию работать довольно сложно. Было огромное количество проблем с Facebook CAPTCHAS, но, похоже, дело идет к необходимости использования аналогичных CAPTCHAS для API третьих сторон. Опять же, тщательно следуйте указаниям на странице плагина WordPress, и он заработает.

После того как авторизация завершена, сложности заканчиваются и в вашем распоряжении оказывается панель управления, позволяющая выбрать необходимую степень интеграции. На выбор предлагается несколько компонентов, каждый из которых имеет несколько вариантов настройки отображения. Одним из самых интересных компонентов данного плагина является возможность агрегирования комментариев записи блога в комментарии Facebook.

Facebook на данный момент является акулой социальных сетей. Широкая аудитория и популярность сопряжены с жалобами и отказами пользователей от нее, однако правда состоит в том, что именно здесь и находится ваша аудитория. Интеграция вашего сайта с Facebook представляет собой компромисс между получением доступа к потенциальным посетителям и следованием условиям и правилам Facebook.

Универсальные данные XML

Теперь, после того как мы рассмотрели самые важные и популярные вопросы интеграции, давайте подумаем над интеграцией простого источника универсальных

данных XML в ваш сайт на WordPress. Это может быть любой источник универсального или специфического контента, имеющийся в вашем распоряжении. В нашем примере мы рассмотрим XML, но это может быть и JSON или любой другой устраивающий вас файловый формат. Вам нужно взять отформатированный XML-фид и принять его, используя PHP-код. В нашем примере мы используем внешний источник данных для того, чтобы передать вашему сайту на WordPress структурную и конфигурационную информацию о контенте. Это может быть также случай, когда вы используете WordPress для разработки внешнего рабочего сайта для вашего заказчика, а он использует внутреннее приложение для управления своей работой. Таким образом, клиент использует внутреннее приложение для получения некоего контента или параметров, а вы разрабатываете интерфейс для передачи этого контента через XML. Ваш сайт на WordPress собирает XML-данные и использует их для настройки сайта. Таким образом, компании приходится управлять данными только в одном месте.

Данный пример будет немного заумным. В большинстве случаев есть множество способов достичь этой цели, и в зависимости от вашей ситуации вы можете выбрать не самый простой или быстрый способ. Вы можете даже подумать: почему бы не воспользоваться функцией управления контентом WordPress для работы с этими данными? Даже с учетом того, что пример нереален, он демонстрирует принципы интеграции, которые могут быть использованы и используются в реальных ситуациях. Данный пример адаптирован и упрощен, но создан на основании используемой в производстве процедуры.

Представьте, что среди ваших клиентов появился продавец зонтов. Директор компании твердо убежден, что для поддержания свежести стиля его сайта необходимо, чтобы цвет фона подходил к цвету недели. Цвет недели, в свою очередь, является цветом того зонта, который выставлен на распродажу на этой неделе. Директор компании выбирает цвет недели каждый понедельник и передает его на точки сбыта. Таким образом все продавцы узнают о том, какой цвет добавить к распродаже.

Директор компании не хочет волноваться о том, что происходит на сайте, но жаждет соответствия цветов. Как было сказано ранее, существует множество способов этого добиться, но когда вы поговорите с системным администратором компании, он скажет вам, что цвет недели содержится в REST API. Здесь мы также сократим наш пример и немного подгоним контекст: соответственно, источник контента и данные реального времени доступны через REST API, а знание о том, как извлечь данные, приведет вас к более широкому выбору внешних источников данных для настройки WordPress. Эта глава посвящена агрегации и потреблению контента, следовательно, у вас есть отличная возможность для применения новых знаний, быстрого захвата потока данных и внесения соответствующих изменений на сайт.

Вот ваш план действий. Сначала вам необходимо создать новую функцию в файле темы `functions.php`, эта функция будет получать данные API, анализировать их и передавать при отображении темы. Приведенный ниже код может легко стать плагином или быть включенным в шаблон файла `header.php`. Для данного примера

он просто должен запускаться при каждой загрузке страницы. (Это неидеальное решение, но данный вопрос будет рассмотрен дальше в следующем разделе.)

В файле `functions.php` вы создаете новую функцию под названием `get_color_of_the_week()`, ее код может выглядеть примерно так:

```
function get_color_of_the_week() {
    $feed = file_get_contents('whatever URL');
    if ($feed) {
        $xml = simplexml_object($feed);
        $color = $xml->color;
    } else {
        $color="white";
    }
    echo $color;
}
```

Данная функция использует PHP-функцию `file_get_contents()` для получения XML из открытого API с точки продаж. Затем данные конвертируются в XML-объект, хотя вы можете добиться тех же результатов с помощью JSON или других форматов данных. Затем задается переменная цвета на основе содержимого динамического XML. Это весьма просто.

Далее вам нужно использовать этот цвет как цвет фона для вашего HTML-элемента `<body>`. В шаблоне `header.php` вы можете сделать следующее:

```
<body <?php body_class(); ?> style="background-color: <?php echo
get_color_of_the_week();?>" >
```

Это позволит использовать встроенный стиль, чтобы перекрыть любой стиль из таблицы стилей CSS. Этот код также должен быть встроенным, так как вы используете PHP для отображения XML-контента. Данный метод почти не отличается от того, как работает встроенный настройщик тем (Theme Customizer) WordPress, рассматриваемый в главе 9.

Метод будет работать только в том случае, если директор компании введет нечто, распознаваемое как HTML-цвет, либо шестнадцатеричный код, подобный `#fffff`, либо название цвета, например *Papaya Whip (Ветвь Папайи)*. Конечно же, в предшествующий код следует добавить проверку и обработку ошибок.

Подумайте о том, что стоит за этим простым примером. Тот же принцип может быть применен к огромному количеству отдельных ситуаций, в том числе настройке сайта через стороннюю систему, как это и было показано. Он может также относиться к контенту, который просматривается и обрабатывается в специальном формате, а также, возможно, к ставкам по кредитам или другим деловым вопросам.

Как вы уже догадались, приведенный в примере метод не очень эффективен. Каждый запрос к каждой странице вызывает функцию REST API для получения информации о цвете фона. В зависимости от загруженности сети или множества других факторов это может оказаться очень долгой процедурой, которая критически

повлияет на время загрузки вашего сайта. Время загрузки является очень важным аспектом для пользователя, это будет рассматривается в главе 12. Кроме того, информация меняется всего раз в неделю, так зачем запрашивать ее при каждой загрузке страницы? Почему бы не запомнить ее или не сохранить на неделю вперед? И именно здесь выходят на сцену временные объекты WordPress.

Временные объекты

Временные объекты WordPress очень похожи на возможности WordPress Options (Свойства WordPress), которые, как и плагины, обсуждаются в главе 8. Основным отличием является то, что временные объекты включают время окончания действия. По своей природе они доступны в течение ограниченного периода, и вы можете думать о них как о самообновляющихся, что делает их идеальными для хранения «ценных» данных в течение определенного периода времени.

Что такое ценные данные? Ценные данные — это данные, на получение или обсчет которых требуется много времени. В предыдущем разделе и в нижеприведенном примере мы рассматриваем то, как получение доступа к API третьей стороны может считаться ценным в зависимости от задержки сети. В любом случае вы не хотите, чтобы время загрузки вашего сайта зависело от доступа к сайту третьей стороны. Однако ценным может быть запрос или расчет, требующий серьезных вычислений, например генерация сложного меню вашего сайта или запрос, охватывающий целую сеть сайта на MultiSite. Вы сохраняете некоторые данные для получения быстрого доступа, что позволит вам держать информацию WordPress «на кончиках пальцев».

Что такое определенный период времени? Здесь есть загвоздка: вы вынуждены балансировать между наличием данных в доступе и их устареванием. В предыдущем разделе рассматривалась ситуация, где период времени, в который данные являются актуальными, определен четко: неделя. В жизни вы можете никогда и не узнать, как часто обновляются данные или как часто их необходимо обновлять. На самом деле, что еще больше осложняет ситуацию, на вашем сайте на WordPress может существовать несколько уровней кэширования. Кэширование более подробно рассматривается в главе 13. Но только представьте масштаб взаимодействия между разными уровнями, где данные могут кэшироваться.

Источник ваших данных может уже содержать кэшированные данные или с задержкой предоставлять внутреннюю информацию API для обработки. Данные ваших временных объектов имеют ограниченный срок кэширования, также при отображении HTML может быть задействован механизм кэширования. В браузере пользователя тоже используется кэширование и может быть даже прокси-кэширование сетевого уровня. Это означает, что существует множество факторов, которые могут повлиять на передачу данных с внешнего источника до браузера посетителя. Эти факторы могут оказаться очень значительными и часто раздражающими при разработке и тестировании интеграции, но они также влияют на период времени, который вы выбираете для хранения временных объектов.

Так что же такое определенный период времени? Каждый раз по-разному. Он зависит от ваших данных, от вашей терпимости к устаревшим данным или требований к актуальности информации, а также от нагрузки на ваш сайт. В предыдущем разделе, когда вы не хотели обращаться к API при каждой загрузке страницы, данные были самими свежими, но само действие накладывало ограничение на время загрузки вашей темы, а также создавало нагрузку на доступ к API-серверу. Временные объекты разработаны для того, чтобы снизить влияние или полностью снять оба заявленных условия. Как разработчик вы должны задавать время истечения срока так, чтобы, с одной стороны, снижать нагрузку на доступ к службе третьей стороны, а с другой — соответствовать ожиданиям на получение новых данных. Вашей целью является вычисление самого длительного периода времени, в течение которого устаревшая или старая информация не будет приводить к получению негативного опыта пользователя. Если говорить человеческим языком, то вам необходимо решить, как сделать ваши данные всегда актуальными, чтобы это не приводило к ошибкам пользователей, обусловленным устаревшими данными.

Воспользуйтесь тем же примером, что и в предыдущем разделе, но добавьте временные объекты для локального кэширования информации. Вы должны внести изменения, включающие хранение временных объектов, в файл `functions.php`. В первую очередь необходимо проверить, есть ли у вас данные с неистекшим сроком. Если ваши временные объекты не существуют или срок их хранения истек, функция `get_transient()` вернет `false`.

```
if (($color = get_transient('color_of_the_week')) === false) {
```

Если при проверке возвращается значение `false`, значит, вам необходимо обратиться к API и получить новые данные. Снова напоминаем вам, что такое кэширование происходит отдельно от кэширования HTML- или PHP-уровня, происходящего на вашем сайте. Описанная проверка напрямую касается конкретной информации. Ниже приведен код, аналогичный рассмотренному в предыдущем примере:

```
$feed = file_get_contents('whatever URL');
if ($feed) {
    $xml = simplexml_object($feed);
    $color = $xml->color;
```

Теперь у вас есть обновленные данные, вам необходимо сохранить их во временном объекте и задать время истечения срока. В основном это работает так же, как и WordPress Options API (API параметров WordPress) с дополнительным параметром истечения срока.

```
set_transient('color_of_the_week', $color, 60*60*24*7);
```

Для удобства чтения ограничение времени указывается в виде формулы. WordPress воспринимает данный параметр в секундах. В нашем примере нам необходимо хранить данные в течение недели, соответственно, расчет будет следующим: 60 секунд × 60 минут × 24 часа × 7 дней, что даст нам неделю в секундах. Вы можете просто ввести значение 604 800 секунды, но время удобнее считывать в формульном виде.

Если мы сложим все приведенное выше вместе, то новая функция с временным кэшированием может выглядеть следующим образом:

```
function get_color_of_the_week() {
    if (($color = get_transient('color_of_the_week')) === false) {
        $feed = file_get_contents('whatever URL');
        if ($feed) {
            $xml = simplexml_object($feed);
            $color = $xml->color;
        } else {
            $color="white";
        }
        set_transient('color_of_the_week', $color, 60*60*24*7);
    }
    echo $color;
}
```

В конце просто используйте этот цвет в вашем HTML, внеся абсолютно такие же изменения, как и в предыдущем разделе. В приведенном фрагменте кода сосредоточена вся магия кэширования. Изменения в вашем шаблоне `header.php` должны выглядеть так:

```
<body <?php body_class(); ?> style="background-color: <?php echo
get_color_of_the_week();?>" >
```

Как видите, добавление пары строчек кода при настройке новой возможности позволяет добавить функцию кэширования, сократить время загрузки вашего сайта, избавиться от необходимости полагаться на стороннее соединение (в течение периода кэширования) и снизить нагрузку при доступе к серверу третьей стороны. Эта весьма простая процедура может существенно повысить эффективность работы вашего сайта с теми данными, которые не должны быть абсолютно новыми.

Реклама

Для продажи товаров или услуг, а также для привлечения внимания потенциальных заказчиков к вашему бизнесу сайт на WordPress будет существенно выгоднее любого центра получения прибыли. С другой стороны, личные дневники или блоги с большим количеством читателей часто имеют высокие цены на рекламу, выступая как сетевой эквивалент объявлений в местных газетах. В данном разделе мы рассмотрим различные аспекты получения прибыли: от настройки полей для рекламы до превращения в спонсируемый торговый сайт.

Если вам вдруг стало интересно, что делает подобный раздел в главе, посвященной агрегации контента, то именно здесь встает вопрос о рекламных сервисах, предоставляемых по подписке. Вы либо берете чужую привлекательную идею рекламы с подходящими ключевыми словами и помещаете ее в поток своего контента, либо размещаете собственную рекламу в чужих ячейках для отображения информации. Также важно учесть разделение аудитории, добавить или создать параллельные

каналы для ваших читателей и подумать о влиянии размещения рекламы на вашем сайте на визуальный и пользовательский опыт.

Монетизация вашего сайта

Существует несколько способов монетизировать ваш сайт на WordPress: размещение рекламы какого-либо крупного рекламного агентства, такого как Google, превращение в представителя сетевой торговой компании, такой как Amazon, оплачивающей переходы по рекламным объявлением, ведущие к увеличению продаж, или же продажа конкретного пространства на сайте заинтересованным спонсорам или рекламодателям. Если вы пойдете по пути коммерциализации, вам также придется принять серьезное решение о передаче части вашего сайта под дизайнерские и пространственные решения третьей стороны. Для личного сайта или блога это в принципе норма. Для коммерческого сайта это, возможно, не то, с чего стоит начинать. Кроме того, есть множество промежуточных вариантов, когда ваше хобби является и вашим делом, — подумайте, например, о сайтах комиксов или о крупных сайтах, посвященных общению.

Пассивная монетизация контента весьма хороша, некоторые популярные сайты зарабатывают на рекламе суммы, достаточные для того, чтобы основать небольшую компанию, хотя среднему блогеру придется довольствоваться редкими просмотрами объявлений. Очень важно иметь разумные представления о том, какую прибыль может принести реклама в вашем блоге, так как ради рекламы вам придется жертвовать эстетикой сайта и его пространством, не имея ни малейшего контроля над тем, какие товары будут рекламироваться и каков будет графический стиль этой рекламы.

Существует несколько моделей оплаты: «оплата за клик» (pay-per-click), когда вы получаете деньги за каждый переход по рекламному объявлению, «оплата за просмотр» и «оплата за день», когда вы получаете доход просто за счет размещения рекламного объявления на вашем сайте в течение указанного периода времени. Google AdSense больше походит на первую модель, так как эта рекламная служба Google устанавливает цену за каждый щелчок по объявлению, основываясь на ключевых словах и контенте сайта, а затем подбирает сочетание доступного рекламодателя и доступного пространства на основе определенной Google рыночной цены и бюджета рекламодателя. Project Wonderful (Проект Чудо) является примером подневной модели оплаты, где вы предлагаете потенциальным рекламодателям заранее определенное пространство, а Project Wonderful проводит постоянный аукцион для выяснения стоимости каждого дня отображения рекламы. Здесь вы получаете доход вне зависимости от того, переходит ли кто-то по объявлениям, размещенным на вашем сайте.

Во всех описанных моделях стоимость места для рекламы на вашем сайте зависит от его популярности и вероятности просмотра объявления или перехода по нему со стороны потенциального покупателя. Если вы можете предложить разнообразный регулярно обновляемый контент, то реклама будет накапливаться на вашем

сайте подобно телевизионной рекламе, идущей поздно ночью по кабельному телевидению. Но если реклама с предложением похудения размещена под вашей записью, то обсуждение полноты тона джазового соло на гитаре заденет ваши чувства и отвлечет от серьезных музыкальных рассуждений. Соответственно, лучше задумайтесь о пассивной монетизации сайта за счет торгового представительства или программ рекомендации, таких как Amazon Associates (представители Amazon), которые платят вам деньги за каждого покупателя, нашедшего их через вас. Кроме того, рекламные сети все больше внимания уделяют исследованию того, куда обращаются пользователи и что они покупают, в результате чего реклама становится все более и более нацеленной на конкретного посетителя сайта. Это значит, что вы видите не те рекламные объявления, которые видит кто-то другой.

Сочетание этих аспектов составляет основу того пути, который проходят пользователи для того, чтобы найти ваш контент и рекламу, размещенную на вашем сайте, а не содержимое, повторно опубликованное через Facebook или агрегированное на других сайтах. Если ваш сайт читают через ленту RSS, убедитесь в том, что менеджер по рекламе размещает рекламу и в ленте RSS.

Размещение рекламы

Размещение рекламы на вашем сайте не отличается от разметки печатной страницы, содержащей смесь редакционного и коммерческого контента, поэтому решите, сколько рекламных блоков вам нужно, где они должны находиться и каких возможных конфликтов контента вы хотите избежать. Это включает поиск как дизайнерских, так и технических решений, поскольку вам необходимо уделять внимание как конечному виду страницы, так и смыслу содержимого, размещаемого рядом с рекламой.

Использование рекламных плагинов

Перед тем как разместить рекламу на своем сайте, вам необходимо создать учетную запись в одном из популярных сервисов, предоставляемых по подписке. В целом это означает, что вам нужно задать имя пользователя, описать свой сайт, продемонстрировать минимальные знания в отношении контента и представить информацию об оплате. Взамен вы получите идентификатор рекламного клиента и зачастую идентификатор ячейки (рекламного блока). Если вы не хотите пользоваться Google AdSense, например, то вам придется создать несколько каналов для ваших рекламных блоков и, возможно, связать разные каналы с разными категориями или частями вашего сайта или использовать отдельный канал для каждого из нескольких управляемых вами сайтов. Идентификатор клиента связан с вашими платежными реквизитами, а каналы описывают различные маршруты для воспроизведения рекламы, которые могут варьироваться по типу контента и размеру, а также иметь разную стоимость перехода.

Существует множество плагинов WordPress для управления вашей рекламой и ее воздействием на ваш сайт. Выбор идеального плагина для вас и вашей рекламной

сети остается на усмотрение посетителя. Просто убедитесь в том, что вы достаточно внимательно прочитали лицензионное соглашение. Некоторые плагины могут выводить несколько ваших рекламных блоков, а потом перемежать их с блоками, полученными из собственных рекламных сетей для покрытия расходов на разработку. Убедитесь, что знаете, на что подписываетесь.

Обычно после установки плагина его настройка является такой же простой, как выбор рекламной службы: вы просто копируете информацию о вашей учетной записи и блоке в панель управления и выбираете особенности отображения, которые управляют размером и формой рекламных ячеек.

Если вы хотите распространить свой контроль на все, что рекламируется на вашем сайте, то не стоит передавать право настройки службе, предоставляемой по подписке, лучше произвести всю настройку самому и затем управлять ею вручную. Есть плагины, которые будут при этом полезны, так как в WordPress есть плагины для всего. Если вы хотите задать возможность спонсирования сайта, когда кто-либо оплачивает поле в заголовке или «подвале» записи или в боковой панели, или же просто хотите продавать рекламные объявления, которые выводятся в разных местах сайта и у каждого из которых своя стоимость, карта и график оплаты, то ручная настройка даст вам наиболее полный контроль за размещением рекламы на вашем сайте.

Размещение рекламы вручную

Рекламные плагины отлично работают в случае, если вы хотите прикрепить рекламное объявление к каждой записи или же хотите, чтобы они выводились в заголовке или «подвале» каждой страницы. Архитектура плагинов, рассматриваемая в главе 8, позволяет отфильтровывать код записи для замены интеллектуального кода шаблоном JavaScript, вызывающим вашу рекламную службу или вставляющим подходящий скрипт в файл с содержимым темы. Если же вы хотите добиться более полного программного контроля над размещением рекламы, такого как более плотная интеграция рекламных блоков в некоторые элементы тем, вам придется вручную редактировать соответствующие файлы и вставлять коды рекламных скриптов.

Вам не понадобится проходить через все этапы преобразования файлов тем для добавления блоков рекламы, так как эта работа является производной от работы по созданию тем, рассматриваемой в главе 9. В целом если вы добавляете рекламный блок в заголовок или «подвал», то его нужно вставлять ниже последнего `<div>` в файле темы, чтобы он не влиял на другие ее элементы. В боковой панели рекламный блок должен быть отдельным элементом списка, а код рекламного блока должен быть окружен HTML-тегами `<li>` и `</li>`.

Процедура размещения рекламных блоков вручную не сильно отличается от добавления значка Facebook или иконки Twitter в вашу боковую панель, так как код блока автоматически генерируется сайтом, управляющим вашей рекламой. Например, если вы используете Google AdSense, то управление и конфигурация страниц в AdSense позволяют вам выбрать форму рекламного объявления (прямоугольную

или квадратную), размер отображаемого блока и используемую цветовую палитру. На конечном этапе конфигурации рекламного блока вы получаете небольшой фрагмент JavaScript, который и добавляете в ваш файл WordPress. В приведенном фрагменте кода идентификаторы клиента и рекламного блока Google были скрыты, но вы можете видеть относительную простоту процесса отображения. Скрипт вызывает службу Google с информацией о точке назначения (информацией о вашем клиенте), размером и идентификатором блока, которые Google использует при обсчете отображения рекламы через те различные «каналы», заданные вами с помощью настроек:

```
<script type="text/javascript"><!--
google_ad_client = "ca-pub-7723xxxxxxxxxxxxxx";
/* кнопка adsense */
google_ad_slot = "747xxxxxxxx";
google_ad_width = 125;
google_ad_height = 125;
//-->
</script>
<script type="text/javascript"
src="http://pagead2.googlesyndication.com/pagead/show_ads.js">
</script>
```

Обратите внимание на то, что переменные, заданные в JavaScript, полностью совпадают с теми величинами, которые вы ввели в плагин AdSense. Еще одним вполне очевидным моментом является то, что каждый раз, когда вы выводите рекламу Google, вы вызываете службу Google, которая решает, какую рекламу разместить, и ждете, пока получите от нее изображение и текст. Обычно это не оказывает серьезного воздействия на работу вашего сайта, за исключением тех случаев, когда Google невероятно загружен и отображение вашего сайта тормозится генерацией рекламного объявления. Ручное редактирование и размещение рекламных блоков других служб являются такими же простыми. Код JavaScript Project Wonderful еще более компактен.

Место размещения рекламных блоков, их форма и стиль взаимозависимы. Если вы собираетесь добавить в боковую панель ряд рекламных блоков, наилучшим вариантом является размещение их в одну колонку и несколько рядов, если, конечно, ширина рекламных блоков подходит под ширину боковой панели, заданную в вашей теме. Идеальным для размещения рекламы в заголовке и «подвале» является использование одного ряда с несколькими ячейками для рекламы или короткой многорядной таблицы по вашему выбору. От размера ваших рекламных полей зависит тип рекламных блоков, которые будут в них размещены: рекламные баннеры, которые занимают всю ширину отображаемой страницы, не подходят для боковой панели и воспринимаются наилучшим образом, когда в блоке размещается только один баннер. Project Wonderful позволяет вам задать несколько видов рекламных блоков для одного сайта, так что если вы хотите подобрать и объединить различные формы, подумайте о широких однорядных блоках для заголовков и «подвалов» и о длинных или высоких таблицах для последовательностей кнопок или квадратных рекламных объявлений в боковых панелях или под записями блога.

Разрешение конфликтов

Конфликты между коммерческим и редакционным контентом, вероятно, начались еще с первых газет, в которых размещалась реклама. В контексте данной книги мы рассмотрим только два вида потенциальных конфликтов: нежелательная для вас реклама и рекламные платформы, не желающие присутствия чужих на их территории.

Нежелательная реклама состоит из неподходящих вам товаров и материала, который вы считаете оскорбительным. Одним из аспектов вопроса является то, как менеджеры по рекламе выбирают рекламу для конкретных ячеек: ключевые слова и сам контекст содержимого должны соотноситься с ключевыми словами, выбранными рекламными агентствами. Если вы, например, пишете о пище, питании и, как следствие, излишнем весе, то не стоит удивляться тому, что под вашими записями начнет появляться реклама борьбы с избыточным весом. Возможно, в этом случае вашей лучшей защитой будет нападение: пишите регулярно, используйте метки и механизмы оптимизации сайта для поисковых механизмов, описанные в главе 12, для оптимального представления вашего контента менеджерам по рекламе.

У всех носителей медийной рекламы есть собственные стандарты ведения бизнеса, такие как невключение в один рекламный блок двух объявлений, поступивших от конкурентов, ведущих одинаковый бизнес. Сетевые рекламные службы имеют свои условия обслуживания на каждом канале, обычно ограничивающие количество рекламных объявлений на странице, типы страниц, на которых может появляться реклама (например, не в письмах электронной почты или не на страницах, в завуалированной форме стимулирующих пользователей их покинуть). Если вы используете Google AdSense, вы имеете право отображать на своем сайте сторонние рекламные объявления, различные по стилю и цвету, что позволяет избежать смешения их с рекламными объявлениями Google. На странице настроек Google AdSense вы можете выбрать цветовую палитру для своих объявлений так, чтобы она отличалась от палитры, использованной в вашей теме, к тому же рекламные объявления, управляемые вашей темой, будут иметь четкие границы и фон. Если ваша тема предусматривает виджет и ячейку, специально отведенные для рекламы, воспользуйтесь ею для размещения резидентной, спонсируемой рекламы или вращающегося баннера. Позвольте AdSense управлять блоками рекламы, привязанными к каждой записи или каждой странице, избегая смешения различных рекламных платформ в одной части отображаемой страницы, ведущего к путанице.

Реклама является неотъемлемой частью публичной визуально воспринимаемой информации еще со времен расцвета телевидения и появления рекламных щитов на трассах. Как администратор сайта, разработчик и создатель контента, вы хотите извлечь выгоду из своей работы, и ваши творческие и редакторские усилия заслуживают потенциального материального вознаграждения.

Личная жизнь и история

Вне зависимости от того, создаете вы личный или рабочий сайт, концентрация вашей сетевой активности на вашем сайте на WordPress выгодно превращает его

в единственный истинный источник вашего присутствия в Сети. Это упрощает для ваших клиентов, семьи или случайных почитателей отслеживание того, что происходит с вами в настоящий момент.

Однако вы, возможно, не хотите этого по целому ряду причин. Одна из этих причин является полной противоположностью того, почему вы можете хотеть агрегировать весь контент социальных сетей, — это ваша личная жизнь. Главной причиной сведения всего в одно является необходимость быть замеченным. Однако то, что делает вас заметным, то и выставляет вас напоказ. И иногда вы не хотите выставлять напоказ свою личную информацию.

Сохранность личной информации должна заботить каждого человека, пишущего в Сети. Google сохраняет все, что может найти, как и Internet Archive (Сетевой архив). В этом нет ничего ужасного, но вы должны помнить, что если что-то стало известным и это было можно найти, то, скорее всего, это можно найти всегда. Кнопки: «А вы уверены?», увы, не существует. Мы процитируем вам старое предупреждение Usenet:

> *Данная программа публикует новости на тысячах компьютеров по всему цивилизованному миру. Рассылка вашего сообщения обойдется Сети в тысячи долларов. Пожалуйста, убедитесь в том, что вы знаете, что делаете. Вы действительно уверены, что хотите сделать это?*

И даже если информация о стоимости давным-давно устарела, сам факт того, что цена когда-то была такой, поможет вам понять, что обратного пути в Интернете нет уже давно. Проще всего каждый раз, когда вы собираетесь что-либо выдать в эфир, задержать дыхание, ведь вы и представить себе не можете его дальнейшую жизнь. Все, что сказано в Сети хоть раз, останется там навечно. Мы даже не знаем, как отреагируют наши дети, узнав, что вся их жизнь с самого рождения подробно описана в Twitter, Facebook и на наших личных сайтах. По прошествии лет фотография маленького тебя, принимающего ванну, может показаться весьма стыдной.

При выкладывании информации вы должны быть осторожны. Все слышали истории о сотрудниках, уволенных из-за Facebook, когда они берут больничные, а затем на своих страницах пишут истории о том, как славно им удалось прогулять и надуть начальника. Размещение неприличных изображений на Flickr тоже может сказываться на вашей репутации весьма долго. За пять минут собеседования с возможным сотрудником и прочтения его резюме очень многие специалисты службы персонала успевают просмотреть информацию о нем в социальных сетях и поисковиках. Это зачастую позволяет узнать подробности, о которых не принято спрашивать на обычном собеседовании. Это такой способ углубленной проверки.

Так или иначе, агрегация всего контента на вашем сайте на WordPress делает его публичным и легкодоступным, а тщательно разработанная сетевая персоналия может положительно сказаться на вашей деловой репутации или продвижении вашей торговой марки. И наконец, сбор всей вашей информации в одной точке

Сети позволяет улучшить восприятие пользователя и расположить его к себе с первого взгляда.

Резюме

В данной главе были рассмотрены некоторые темы, касающиеся того, как дать вашему сайту WordPress возможность участвовать в жизни всей Сети, включая продвижение вашего контента в сторонних источниках и привлечение внешнего контента на ваш сайт. Сторонний контент может представлять собой копию или комментарий, а также службу, обогащающую возможности ваших клиентов или позволяющую изменить информацию на вашем сайте. Сторонний контент был также представлен в форме рекламы, которая может способствовать монетизации вашего сайта. Кроме того, в данной главе обсуждались последствия рассеивания вашей аудитории по различным местам в Сети. В следующей главе мы сосредоточимся на том, как на основе вашего контента выстроить позитивное взаимодействие с пользователем.

12 Взаимодействие с пользователем

В этой главе:

- Принципы взаимодействия с пользователем
- Изучение преимуществ простоты использования и ее проверки
- Оптимизация сайта для поисковых систем
- Усовершенствование встроенного поиска WordPress

Несколько предыдущих глав были посвящены созданию и представлению вашего великолепного контента пользователю. На самом деле эти главы были посвящены вашим собственным целям, тому, каким бы вы хотели видеть свой сайт, как бы вы хотели, чтобы он работал, и тому, какое содержимое вы представляете. Но ведь речь идет не только о вас. А как насчет посетителей вашего сайта? Вот тут и становится актуальным взаимодействие с пользователем.

До настоящего момента все внимание было сосредоточено на том, как создавать сайт и управлять контентом. Теперь вопрос заключается в том, будет ли сайт привлекать и удерживать внимание зрителей (пользователей) благодаря механизмам и решениям, которые вы задействуете. Так как вы имеете дело с людьми, их уникальное восприятие — весьма неопределенная штука. Это ситуация, когда о вкусах и правда не спорят.

Опыт пользователя действительно состоит из чего-то большего, чем просто впечатления человека, сидящего где-то в Поволжье перед открытым окном браузера. Косвенно поисковые машины, поисковые системы и потребители услуг, такие как RSS-боты, также являются вашими пользователями. Ваш сайт должен быть собран и структурирован так, чтобы все группы пользователей имели преимущества и получали уникальный опыт.

Принципы взимодействия с пользователем

Взаимодействие с пользователем — открытая тема, каждый видит ее немного по-своему. Но все же имеются некоторые ценные указания. Часть этих указаний основана на здравом смысле или хотя бы кажется таковой. Эти указания нацелены на установление равновесия между тем, что эстетически приятно, и тем, что практично. Это простые и проверенные советы, которым стоит следовать. Люди непостоянны, поэтому и любые указания должны меняться для того, чтобы удовлетворять их потребности.

Вот несколько простых вопросов:

- Есть ли у моего сайта единый стиль?
- Дизайн полезен или вреден?
- Легко ли найти и получить доступ к содержимому?
- Хорошо ли структурирован контент?
- Загружается ли мой сайт достаточно быстро (в разумных пределах)?

Данный список позволяет выделить основные моменты восприятия пользователя. То, как вы сможете их использовать или в каком сочетании, как раз и описано в данной главе. Нижеследующие разделы внесут в заявленные темы некоторую ясность.

Единая навигация

Единый стиль. В наше дни это абсолютно просто, так как практически невозможно не добиться единого стиля при использовании тем WordPress. Вы ведь хотите, чтобы ваши посетители были в курсе того, что они постоянно пользуются вашим сайтом, вне зависимости от тех методов, которыми вы этого добьетесь? Это значит, что вашему сайту необходим единый стиль, включающий «шапку» и постоянную глобальную навигацию.

Это не значит, что отдельные разделы не могут иметь уникального стиля, но и они должны быть частью целого. Вашего пользователя может сильно дезориентировать ситуация, когда он будет читать некое содержимое на одной странице, а затем щелчком мыши будет направлен на следующую страницу с абсолютно иным стилем. Посетителям покажется, что они покинули ваш сайт, и они не отдадут должное всему великолепному контенту, созданному вами.

Кроме того, каждая страница вашего сайта должна иметь постоянную глобальную навигацию. Постоянство означает, что страница не меняется и не перемещается. Посетители должны иметь возможность исследовать содержимое сайта без страха потеряться в нем. Это может прозвучать глупо для специалиста вашего уровня, однако у среднестатистического пользователя несколько другие отношения с технологиями. Глобальная навигация — спасательная шлюпка для ваших пользователей, позволяющая им возвращаться в то место, с которого они начали.

Качественная глобальная навигация также сообщает пользователям, где именно на сайте они находятся. Вы можете добиться особого успеха, создав «активный» пункт меню и выделив его подсветкой или любым другим способом, отличающим от других пунктов в рамках глобальной навигации. Это позволяет пользователю посмотреть на навигационную панель и сразу увидеть, где на вашем сайте он находится относительно других разделов или страниц. Встроенная система меню WordPress делает это автоматически; вы автоматически получаете класс `current_menu_item` в меню, активном на текущий момент. Вот пример кода HTML:

```
<li id="menu-item-44" class="menu-item menu-item-type-post_type
    menu-item-object-page current-menu-item page_item
    page-item-42 current_page_item menu-item-44">
<a title="Register Me" href="/register-me/">
  Register Me
 </a>
</li>
```

В качестве альтернативы вы можете создать навигацию самостоятельно с помощью встроенной в WordPress функции `is_page()`, которая позволяет добиться аналогичных результатов (код PHP в файле шаблона):

```
<li class="benefits
 <?php if(is_page('benefits')){ echo "current_menu_item";}?>">
  <a title="benefits" href="/benefits/">
   Benefits
  </a>
</li>
```

В обоих случаях класс `current_menu_item` позволит отделить желаемый пункт меню от прочих с помощью заданного стиля CSS. Мы (авторы) твердо верим в то, что функции обратной связи с пользователем являются ключевыми элементами. Во-первых, наведение курсора на элемент должно включать некоторое подобие обратной связи помимо превращения курсора в «ладошку». Обычно такое переключение подразумевает подсветку или затемнение шрифта, фона или кромки. Во-вторых, раздел навигации, активный на текущий момент, должен быть выделен аналогичным образом, но отличаться от прочих. Два этих принципа вместе позволяют создать глобальную навигацию, визуально представляющую достаточное количество информации о том, где находится пользователь относительно остальной части сайта и куда еще он может перейти для получения дополнительной информации. CSS, адаптированная из темы Twenty Eleven, может быть следующей:

```
#access li:hover > a,
#access ul ul :hover > a,
#access a:focus {
 background: #5B84BA;
}
#access .current-menu-item > a,
#access .current-menu-ancestor > a,
#access .current_page_item > a,
#access .current_page_ancestor > a {
 font-weight: bold;
```

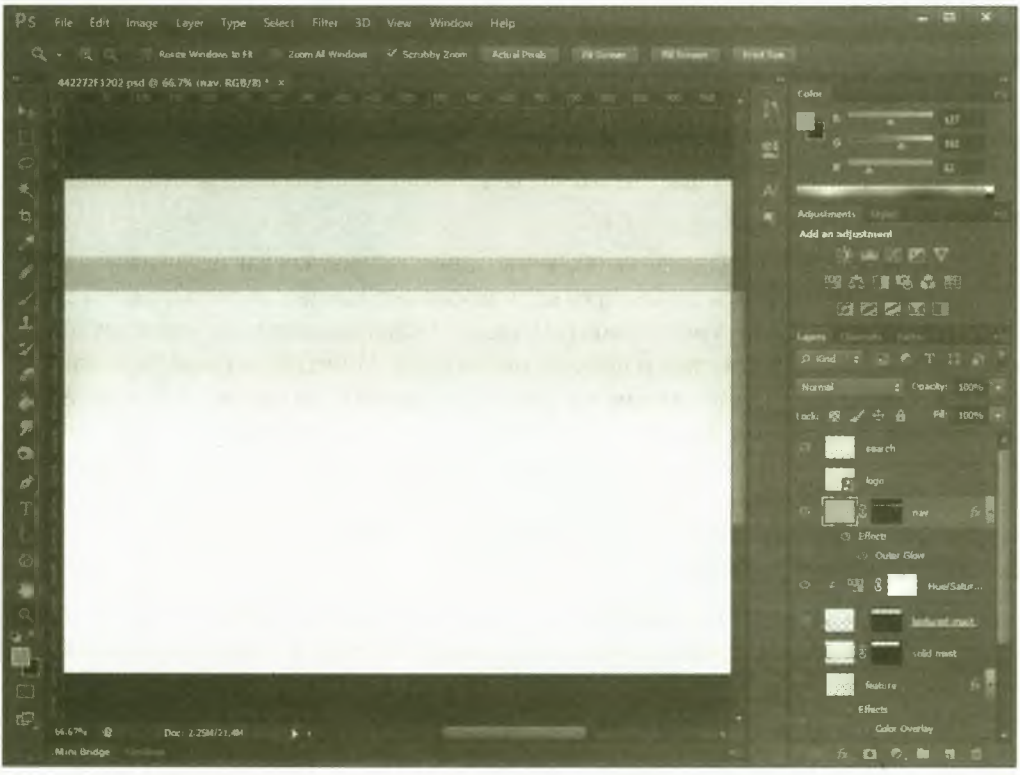


Рис. 12.2. Макет, разрабатываемый в Photoshop

и сами эти слои были рассмотрены и признаны избыточными. Избавление от них создало более сильную композицию.

Упрощение поиска контента

При наличии успешного сайта вы однажды накопите довольно большой объем контента. И очень часто посетитель вашего сайта не сможет в своей голове категоризировать или организовать контент так, как это сделали вы. Соответственно должно быть множество путей, позволяющих добраться до всего содержимого сайта. Это повысит вероятность того, что пользователям удастся найти именно то, что они ищут. Это отличная причина для создания рубрик, меток и шаблонов хронологических архивов. Наличие таких шаблонов связано с обращением к трем самым распространенным способам, которые люди используют, чтобы запомнить, где что-то было. Кроме того, это послужит вам инструментом для продвижения более активного взаимодействия с контентом, его потребления и поможет продемонстрировать ваши взгляды (как творца) на хранение контента.

WordPress с самого начала помогает реализовать данную стратегию. Во-первых, на вашем сайте должна присутствовать постоянная глобальная навигация, о чем мы уже говорили. WordPress всей своей структурой поддерживает ее использование,

Элементы графического дизайна

Честно спросим, а являются ли визуальные элементы вашего сайта полезными или же они отталкивают пользователя? Укрепляет ли тема представление о выбранном вами образе или же отталкивает от него? Это еще одна из тем, которые открыты для индивидуальной интерпретации. Фотографии и цветовая гамма запускают различные субъективные реакции у разных людей, но общее впечатление, производимое вашим сайтом, должно в любом случае соответствовать основному контенту.

Например, бизнес-сайт не должен быть выполнен в ярко-розовом цвете, если, конечно, создатель хочет, чтобы сайт воспринимали всерьез, или же этот сайт продает игрушки или жевательную резинку. На другом полюсе находится в последнее время некоторое неприятие использования розового цвета для оформления сайтов, связанных с вопросами женского здоровья, так как чрезмерное и необдуманное использование этого цвета нивелирует эффект и впечатление от его воздействия. Графический стиль должен представлять марку и те ее преимущества, которые вы хотите передать посредством своего сайта.

Цвета пробуждают различные чувства. Синий вселяет надежду, именно поэтому он и преобладает в деловых и финансовых логотипах. Оранжевый намекает на новые технологии и часто используется в сфере телекоммуникаций. Цветовые решения для торговых марок — отдельная самостоятельная тема, но вы должны понимать, что розовые пони — не лучший способ продвинуть на рынке ваш банк, а черепа и скрещенные кости не помогут раскрутить сайт детской мебели.

Данная тема неоднозначна. Она эмоционально окрашена, и очень часто при ее обсуждении маркетинг побеждает здравый смысл. Например, весьма вероятно, что многие из вас работали с заказчиками, абсолютно убежденными, что каждый новый элемент, добавленный на главную страницу, будет на ней самым важным. Это значило, что каждый элемент дизайна был очень большим, мигающим и делал всю страницу просто тошнотворной. А ведь все банально, как в рекламе: все эти новые и улучшенные формулы — один и тот же стиральный порошок.

Одной из теорий дизайна, которая может помочь вам при создании нового макета, является детальная проработка композиции. Выстраивайте слои элементов, работающие на вашу конечную цель. Когда вы уже решите, что финальный макет вас устраивает, удалите один элемент. Через некоторое время верните его. Используйте этот подход, являющийся вариантом подхода «чем меньше, тем больше».

Возьмите, например, макет, разработанный в Adobe Photoshop, как это показано на рис. 12.2. В слоях палитры справа вы увидите, что все компоненты макета разбиты на группы по слоям. При создании этого макета каждый графический элемент был создан с использованием описанного метода, что позволяет легко заменять и перемещать элементы, не нарушая структуру остальных слоев. Вы также увидите, что некоторые группы слоев на данный момент отключены, — около них будет отсутствовать иконка «глаз». При разработке данного макета и графические элементы,

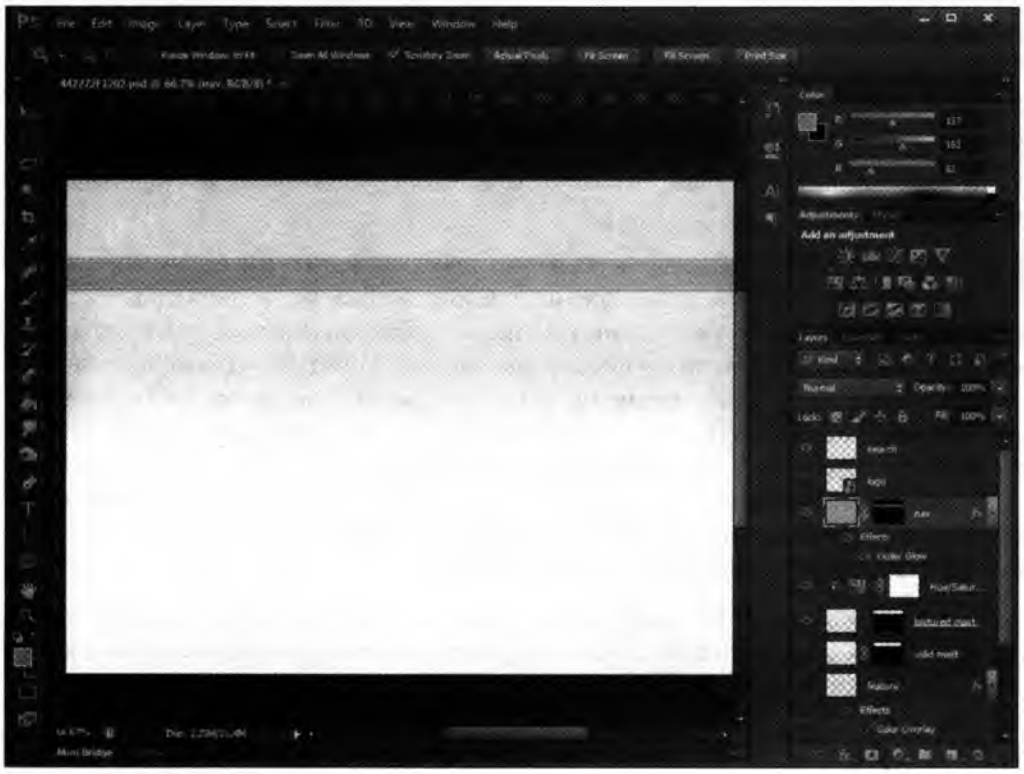


Рис. 12.2. Макет, разрабатываемый в Photoshop

и сами эти слои были рассмотрены и признаны избыточными. Избавление от них создало более сильную композицию.

Упрощение поиска контента

При наличии успешного сайта вы однажды накопите довольно большой объем контента. И очень часто посетитель вашего сайта не сможет в своей голове категоризировать или организовать контент так, как это сделали вы. Соответственно должно быть множество путей, позволяющих добраться до всего содержимого сайта. Это повысит вероятность того, что пользователям удастся найти именно то, что они ищут. Это отличная причина для создания рубрик, меток и шаблонов хронологических архивов. Наличие таких шаблонов связано с обращением к трем самым распространенным способам, которые люди используют, чтобы запомнить, где что-то было. Кроме того, это послужит вам инструментом для продвижения более активного взаимодействия с контентом, его потребления и поможет продемонстрировать ваши взгляды (как творца) на хранение контента.

WordPress с самого начала помогает реализовать данную стратегию. Во-первых, на вашем сайте должна присутствовать постоянная глобальная навигация, о чем мы уже говорили. WordPress всей своей структурой поддерживает ее использование,

но само решение вопроса о создании и организации навигации остается на ваше усмотрение.

Во-вторых, WordPress поставляется вам со встроенной функцией поиска. И хотя эту функцию можно улучшить, как мы расскажем далее в данной главе, наличие поиска в любом случае лучше его отсутствия. Поиску способствует использование меток.

В-третьих, WordPress предлагает множество альтернативных взглядов на контент: либо через особые шаблоны, либо через использование файла шаблона главной страницы, установленного по умолчанию (см. главу 9 «Разработка тем» для получения более подробной информации о файлах шаблонов). WordPress также предлагает организовывать ваше содержимое по датам, рубрикам, заголовкам или авторам или же еще несколькими иными способами. Творческий подход к использованию этих шаблонов и других механизмов произвольных циклов создает еще одну грань вашего контента. Основным моментом здесь является то, что этот метод позволяет обслуживать дублирующийся контент, который игнорируется поисковыми машинами, но эта проблема будет рассмотрена далее.

В-четвертых, существует множество плагинов для связанных записей. Добавление списка связанных записей в нижнюю часть записи является еще одним способом, обеспечивающим более глубокое погружение в ваш контент. Особенно эффективен этот метод, если текущий контент уже завладел вниманием вашего посетителя. Предложение аналогичного содержимого — отличная идея, к тому же это способ предоставить по заданной теме больше информации, которая может быть полезна пользователю.

Время загрузки сайта

В те времена, когда установление соединения с Интернетом происходило в основном по телефонной линии, сетевые разработчики были чрезвычайно озабочены размером страниц и временем их загрузки. Во времена же широкополосного доступа разработчики больше не волнуются о времени загрузки или обновления уже загруженного сайта. Файлы CSS процветают, вместо слияния имеющихся стилей добавляются новые стили и селекторы. Библиотеки AJAX и JavaScript (притом иногда более одной JavaScript-библиотеки) встраиваются просто для достижения ярких визуальных эффектов. iFrames, сетевые сервисы и другие компоненты, разработанные третьими сторонами, заставляют HTML-документы прямо-таки разбухать. Многочисленные запросы к базам данных для сбора информации замедляют работу страницы на уровне сервера.

Кроме того, разработка новых стилей требует использовать все больше CSS, изображений и прочих эффектов, ускоряющих переполнение широкополосных каналов. Это проблема как новых разработок, так и поддержания наследуемых стилей.

Так является ли время загрузки все еще важным аспектом? Естественно, является. Сам факт того, что скорость получения доступа к сетевым ресурсам возросла,

не означает, что следует игнорировать необходимость оптимизации кода. Тем не менее на самых ранних стадиях процесса проводить оптимизацию не стоит. Досрочная оптимизация замедляет разработку и ввод в эксплуатацию вашего сайта. Существует строгое равновесие между выполнением работ, выпуском готового товара и оптимизацией его в целях обеспечения успешного запуска. Время загрузки страницы должно определяться уже при разработке вашего сайта. Правильно оптимизированный сайт загружается намного быстрее, чем сайт, собранный кем-то, не понимающим последствий своих действий.

Данная проблема может оказаться очень сложной: только задумайтесь обо всех аспектах, влияющих на время загрузки. Есть вполне очевидные, те, с которыми вы давно знакомы, например количество и размер изображений, количество используемых библиотек JavaScript и эффектов, создаваемых этими библиотеками. Примите также во внимание интеграцию с другими сайтами, такую как использование избыточного количества иконок Facebook, позволяющих обновлять статус и оценивать контент, или же большое количество изображений, размещенных посредством ссылок на посторонние хранилища. Что произойдет, если все эти удаленные сайты будут отвечать очень медленно или, хуже того, не будут отвечать совсем? Может ли время отклика вашего сайта пострадать из-за чего-то, что вы не в силах контролировать? Обдумайте древо связей, влияющих на эффективность, которое создается при обращении на другие сайты. Это не означает, что вы не должны обращаться к ним вовсе, однако необходимо осознать их влияние на работу вашего сайта.

Firebug продолжает быть превосходным инструментом для работы по оптимизации и повышению пропускной способности сети для вашего сайта. Инструменты разработчика Google Chrome — тоже хороший вариант, пользующийся популярностью. Кроме того, у Yahoo! и Google есть специальные расширения для Firebug и Chrome Developer Tools (Инструменты разработчика Chrome) для повышения скорости работы вашей страницы: YSlow (http://developer.yahoo.com/yslow/) и Page Speed (http://code.google.com/speed/page-speed/) соответственно.

Тормозом работы YSlow и Page Speed является то, что они предоставляются сетевыми монополистами. Через эти сайты за час проходит больше трафика, чем вы расходуете за год. Проблемы и вопросы увеличения скорости, актуальные для них, это не те проблемы, которые актуальны для вас. YSlow всегда рекомендует использовать сеть распределения контента (CDN, Content Delivery Network). Естественно, CDN распределит все ваши данные по географически сложной сети серверов, что позволит увеличить надежность и сократить время ожидания, но действительно ли это нужно вашему сайту? Действительно ли вам нужно платить за это? Выбор за разработчиком, но всегда помните, что ваш сайт далеко не того же масштаба, что Yahoo! или Google.

Вам необходимо сортировать и решать проблемы в области времени загрузки сайта. Вот небольшой список вопросов для рассмотрения, начинающийся с самых базовых:

- Оптимизируйте графику, подберите подходящие разрешение, глубину цвета и формат. Не забудьте о более высоком разрешении для устройств компании Apple и других мобильных устройств.

- Стандартизируйте свою библиотеку JavaScript и пользуйтесь только ею. Оцените преимущества архивирования и минимизации JavaScript и CSS. Впрочем, эти усилия могут и не привести к сокращению времени загрузки страницы.
- Оцените количество существующих внешних ссылок — от переадресации на изображения в удаленных хранилищах до кнопок Facebook с возможностью обновления статуса. Подумайте об использовании временных объектов, описанных в главе 11.
- Разумно относитесь к производительности баз данных MySQL на вашем хостинге. Так как любая страница или запись с отсрочкой включает запросы к базе данных, убедитесь, что вы не перегружаете ваш ведущий узел. Плагины, сохраняющие контент в базе данных, дают вам гибкость, однако увеличивают массу запросов к базе данных, когда вы производите вывод страниц. Здесь также могут оказаться полезны временные объекты.
- Кэширование данных вывода может оказаться для вас подходящим решением, оно более подробно описано в главе 13 «Статистика, масштабируемость, безопасность и спам». Вам придется оценить все возможности внедрения и подобрать решение, соответствующее требованиям к масштабу вашего сайта и устраняющее препятствия к его эксплуатации.

Использование JavaScript

Еще один полезный совет по использованию JavaScript в дизайне вашего сайта: вам время от времени будет хотеться создать навигацию по сайту (или другой элемент стиля) полностью на основе великолепного плагина jQuery. JavaScript поможет добиться действительно запоминающегося эффекта, но он не должен становиться ядром вашего дизайна. Не стоит использовать эффекты jQuery как обсыпку праздничного торта. Вам необходима надежная основа для того, чтобы сайт продолжал функционировать и радовал глаз, даже если блестки JavaScript померкнут. Сайт нужно строить снизу вверх, а лоск наводить на уже работающую версию. Поймите, что каждая дополнительная библиотека JavaScript или яркий эффект, которые вы добавляете, увеличивают время загрузки сайта для конечного пользователя. Проверьте, добавляет ли эффект изюминку на сайт или просто кажется вам симпатичным.

Кроме того, убедитесь, что в случае отказа JavaScript сайт продолжает выглядеть привлекательно. Вам как бы нужно сделать торт вкусным и без обсыпки. Если работа некоторых функций вашего сайта основывается только на JavaScript и другого пути для работы этих функций нет, то однажды сайт может оказаться недоступным. Учтите, что у какого-то числа пользователей JavaScript не будет включен, а возможно, не будет им даже доступен, и у вашего сайта на этот случай должна оказаться элегантная «голая» версия для поддержки пользователей.

В любом случае, необходимы некоторые усилия или же перераспределить визуальные элементы для сокращения времени загрузки страницы. Стоит соизмерять

необходимые ресурсы с их положительным воздействием на восприятие пользователя.

Простота использования и ее проверка

Ваш заказчик может не быть конечным пользователем. Более того, ваши заказчики не знают, чего в действительности хотят пользователи. С точки зрения создания контента ни разработчики, ни редакторы не будут точно знать, чего хотят их читатели (пользователи), если не будет работать механизм обратной связи — тестирование. Веб-дизайн — одна из тех странных профессий, где каждому кажется, что он знает, как лучше. Подумайте снова о том заказчике, который хотел, чтобы каждый элемент был самым важным на странице, и в итоге создал каскад мигающих значков.

Ваши заказчики в основном считают, что знают о желаниях своих пользователей все, просто потому, что знают, чего бы хотели они сами, заходя на сайт определенного рода, сайт, который вы разрабатываете. Иногда это срабатывает. Однако чаще всего у ваших заказчиков весьма индивидуальное представление о вопросе, и притом весьма необъективное, так как у пользователей такого представления нет вовсе.

Если вы серьезно нацелены на создание качественного пользовательского интерфейса, начинайте тестирование как можно раньше и проводите его как можно чаще. Вам придется решать, что именно вы собираетесь тестировать, а это, в принципе, зависит от целей, стоящих перед вашим сайтом. Например, сайт для электронной коммерции обычно нацелен на продажу продукции.

ПРИМЕЧАНИЕ

Данная история использовалась в качестве примера для проверки простоты использования в течение довольно долгого времени, но она представляет собой занимательный рассказ о том, как изменение одной кнопки принесло в кассу 300 миллионов долларов http://www.uie.com/articles/three_hund_million_button/.

Как же вам применить этот подход к тестированию своего сайта? A/B-тестирование — неплохой способ проверить, что работает в условиях реального мира. В случае с тестированием «А или Б» у вас будут две различные версии рабочей сетевой страницы. Ваш сайт будет случайным образом демонстрировать ту или иную версию каждому посетителю. Этот процесс требует некоторых фокусов с кодом и является простым способом проверки юзабилити на большой аудитории, но для этого ваш сайт должен быть рабочим. Из результатов теста вы сможете понять, какой из вариантов страницы работает лучше. Это, в некотором роде пример того, как проверяет простоту использования Google — он на лету модифицирует свои сервисы и смотрит, какие из них генерируют реальный трафик. Если вы собираетесь попробовать A/B-тестирование вашего сайта на WordPress, есть несколько плагинов, упрощающих процесс. Некоторые из них даже работают с Content Experiments (Эксперименты с контентом) Google Analytics при получении обратных ссылок.

Еще одной возможностью для вас будет помощь семьи и друзей или просьба о помощи, опубликованная вTwitter. Это называется *краудсорсингом*.

Любой способ тестирования лучше, чем отсутствие такового. Наличие второй пары глаз — отличная идея. Вам совершенно не обязательно соглашаться с результатами тестирования и вносить изменения, предлагаемые пользователями, однако стоит принимать их во внимание. И опять же, вторая пара глаз и новый взгляд человека, не знакомого с сайтом так близко, как вы, — весьма неплохая идея.

Если в ваш бюджет не заложена проверка простоты использования, обратитесь к членам вашей семье и понаблюдайте, как они взаимодействуют с сайтом. Рассмотрение того, как они находят информацию, позволит вам идентифицировать места, где ваш дизайн может быть усовершенствован, а выслушивание их комментариев даст вам шанс увидеть хорошие и плохие стороны стиля в целом. Иначе говоря, ваша семья и друзья представляют собой хорошую команду для тестирования и обладают среднестатистическими навыками работы с компьютером.

Кроме того, вы можете обратиться за помощью незнакомых людей через социальные сети. Однако полученные вами результаты окажутся непредсказуемо разнообразными. В целом люди будут вежливы и дадут вам некоторое количество положительных или отрицательных комментариев, возможно вам даже удастся получить полноценный развернутый отзыв. Увы, такая процедура занимает у среднего пользователя слишком много времени.

Возможно, вы сможете получить более детальные отклики от подписчиков местной группы пользователей WordPress. В нее обычно входят ваши коллеги-разработчики, в отзывах которых можно найти разумное зерно. Некоторые группы более неоднородны по соотношению разработчиков и пользователей, особенно если поиграть с ними в «покажи и назови». Такие группы предоставляют неплохую возможность получить отзыв и провести тестирование.

Если же ваш бюджет позволяет, то обратитесь к сайтам, предоставляющим услуги агентов пользователя при тестировании. Обычно при использовании таких сервисов вы подаете заявку или представляете свой сайт и ставите некоторые цели, которых должен достигнуть пользователь. Вы также можете выбрать целевой уровень владения компьютером. Далее сервис обращается к своим агентам, являющимся средними пользователями, имеющими дома некоторое программное обеспечение, такие пользователи предоставить вам запись своих попыток достигнуть поставленных вами целей.

Мы воспользовались одним из таких сервисов для проверки полностью нового внешнего интерфейса одного из наших ключевых сетевых приложений (не относящегося к WordPress). Полученные записи позволили нам увидеть взаимодействие пользователя с сайтом, а также содержали звукозаписи комментариев пользователей. Некоторые комментарии были совершенно ясными, другие же требовали интерпретации эмоций, ворчания и хмыканья. В итоге тестирование простоты использования позволило нам выявить несколько аспектов, требующих упрощения и повышения эффективности подкрепления некоторых действий,

а необходимые изменения были внесены уже с помощью более опытных внутренних фокус-групп.

Команда WordPress провела тестирование простоты использования панели инструментов для нескольких последних релизов. Похоже, что панель инструментов администратора практически через раз проходила серьезную перестройку. WordPress, в силу того что он разрабатывается целым сообществом, представляет собой удивительный пример краудсорсинга как в плане разработки, так и в плане тестирования простоты использования. Такая пользовательская проверка ориентирована на те функции WordPress, которые используются наиболее часто, и именно она напрямую привела к разработке QuickPress и других функций. Такое тестирование большим числом пользователей также привело к созданию очень функциональной панели управления.

Пока вы читаете эти строки, тестирование пользователями проходит свой длинный путь в направлении усовершенствования структуры и дизайна вашего сайта. Очень часто этому тестированию не уделяют необходимого внимания, так как разработчики — умные люди и зачастую знают все наперед, однако с учетом того, что вы-то знаете свой сайт досконально, свежий взгляд может сделать его лучше, особенно если вы прислушаетесь к некоторым советам.

Структурирование информации

То, как организован ваш сайт, критически важно для пользователей и поисковых машин. В целом WordPress неплохо справляется с организацией вашего контента. В конце концов, это и должно быть основной задачей системы управления контентом. Тем не менее вам стоит немного задуматься об общей структуре вашего сайта.

Одной из первых вещей, о которых вы попросите заказчика, планирующего изменить дизайн сайта или разработать новый, является создание иерархии страниц или контента будущего сайта. Такая просьба заставит заказчика взглянуть на структуру и организацию сайта в целом с высоты 3000 метров. То, к какому типу контента относится каждый элемент, также помогает в полноценной структуризации потока информации на сайте. Используя подобные иерархии, разработчики получают возможность конкретизировать информационную архитектуру рубрик записей, страниц, родительских страниц, привести ее в соответствие с представлениями заказчика и облегчить себе работу по созданию сайта. Это также позволяет заказчику увидеть разметку сайта уже на макете («рыбе») и внести в структуру необходимые изменения на начальном этапе работы.

Давным-давно существовало «золотое правило» интернет-сайтов: ни одна страница не должна быть дальше трех кликов мыши. Это было в те далекие времена, когда миром доступа в Сеть управляло соединение по телефонной линии. Несмотря на то что мы не являемся поклонниками громоздких сайтов, мы не уверены, что это правило действует и по сей день. Не то чтобы поле внимания посетителя стало значительно шире, мы бы даже сказали: оно стало уже. И само собой, практически

везде сейчас присутствует широкополосный доступ, так что вес страницы — это не то, что в данный момент является для вас определяющим.

Короче говоря, поиск в большой степени заменил вертикальную навигацию. Мы считаем, что люди не отправляются на главную страницу сайта и не продираются через глобальную навигацию, чтобы найти какую-то тему, статью или нужный товар. Чаще всего они обращаются к поисковику. Поисковик предоставляет им прямую ссылку на конкретную страницу или на максимально близкую страницу из возможных вне зависимости от той глубины, на которой она находится на сайте.

Несмотря на то что мы все еще отдаем предпочтение дизайнерскому правилу «не дальше трех кликов мыши», мы занимаемся этим лишь потому, что оно соответствует принципу «чем проще, тем лучше», и делаем использование вашего сайта более простым. Тем не менее не следует считать это правило самым проверенным и надежным: сайты весьма сложны и включают намного более многоуровневый контент, чем тот, который существовал во времена использования этого правила. Вложение усилий в то, чтобы упростить обнаружение вашего содержимого при использовании поисковиков, и структуризация самого контента в соответствии с правилом трех кликов смогут вместе сделать взаимодействие с пользователем более эффективным.

Также сейчас идеальное время для оценки заголовков отдельных страниц и разделов. Вот печальная правда веб-дизайна: ваш контент в действительности никто не читает. В 2006 году исследование, организованное Якобом Нильсеном (http://www.useit.com/alertbox/reading_pattern.html), показало, что посетители просматривают содержимое сайта очень быстро, по диагонали, их взгляд движется вниз по левой стороне страницы и останавливается на заголовках в поисках нужного содержимого. 2006 год кажется уже далеким прошлым для мира высоких технологий, однако именно с тех пор упомянутое исследование постоянно приводится в цитатах.

Опять-же, утверждение небезосновательно. Очевидно: люди читают контент и статьи сайтов, иначе не было бы смысла их создавать. Но если вы пытаетесь привлечь посетителей и заставить их «лазать» по вашему сайту, что же вам стоит вынести из описанного исследования?

Заголовки важны. Заголовки должны быть краткими и дескриптивными. Ваш контент должен начинаться с наиболее важной и привлекательной информации и уже потом уходить вглубь вопроса. Заголовки должны быть правильно отформатированными, что позволит использовать разные уровни HTML (мы позже расскажем об этом более подробно). Заголовки должны содержать слова-действия. Они также должны быть интересными и стимулировать пользователей читать ваш контент, если это, конечно, ваша цель. Если пользователь просматривает сайт, то наличие в заголовках действий и описаний позволит ему составить общее представление о странице, поможет в поиске необходимой информации и, возможно, заставит прочитать остальные разделы.

Например, какая структура содержания кажется вам более информативной и интересной? Эта:

- Как пользоваться WordPress
  - Введение
  - Технология
    - Программное обеспечение
    - Оборудование
  - С чего начать

Или эта:

- Публикация вашего контента в Интернете с помощью WordPress
  - Какие шаги необходимо предпринять?
  - Что мне нужно?
    - Установка приложений
    - Настройка сервера
  - Начало публикации

Как вы видите, первый вариант содержания позволяет вам получить общее представление о сайте. Однако второй вариант намного более интересен и привлекает ваше внимание к задачам, требующим непосредственного действия. Вам также видна структура и логика.

Помните, как в школьные годы вам нужно было писать контент с заголовками разного уровня? Теперь перед вами стоит та же задача. Ваш контент должен иметь структуру, заголовки и опорные параграфы. Если заголовок достаточно заинтригует посетителя, то он прочитает опорные параграфы. Если нет, то он просмотрит текст до следующего заголовка. Забавно, что школа все же научила вас чему-то, что действительно пригодится в жизни, не правда ли?

Как сделать ваш сайт легко обнаруживаемым

Оптимизация сайта в поисковых системах (SEO) — это способ заставить поисковые машины находить ваш сайт. Одним из основных путей к этой цели является использование структуры постоянных ссылок в WordPress. Эти постоянные ссылки, оптимизированные для поисковых механизмов, являются одним из главных факторов, они отображаются на страницах с результатами поиска всех поисковых машин. Абсолютно необходимо сделать их информативными и дескриптивными.

Увы, в исходной версии WordPress структура адресов (URL) использует формат строки запросов с идентификатором в конце (`http://example.com/?p=100`). Это

сделано из соображений совместимости и является функцией по умолчанию, так как гарантированно работает на различных платформах и серверах.

Если бы у вас была возможность выбрать страницу с результатами поиска между двумя адресами: http://example.com/?p=42 и http://example.com/this-is-the-information-you-want, какой вариант вы бы выбрали? Ответ очевиден. При использовании второго варианта посетитель или потенциальный посетитель получает хотя бы базовое представление о том, что ожидает его на сайте. Данный дескриптивный адрес эффективен в отношении поисковых механизмов и переходов по Сети, так как умный пользователь умеет заглядывать в панель статусов своего браузера в поисках нужного сайта. По этой причине мы настоятельно рекомендуем с самого начала, еще при создании сайта на WordPress, изменить структуру архивных ссылок, как это показано на рис. 12.3. Само собой, вам необходимо иметь платформу, которая такие ссылки поддерживает.

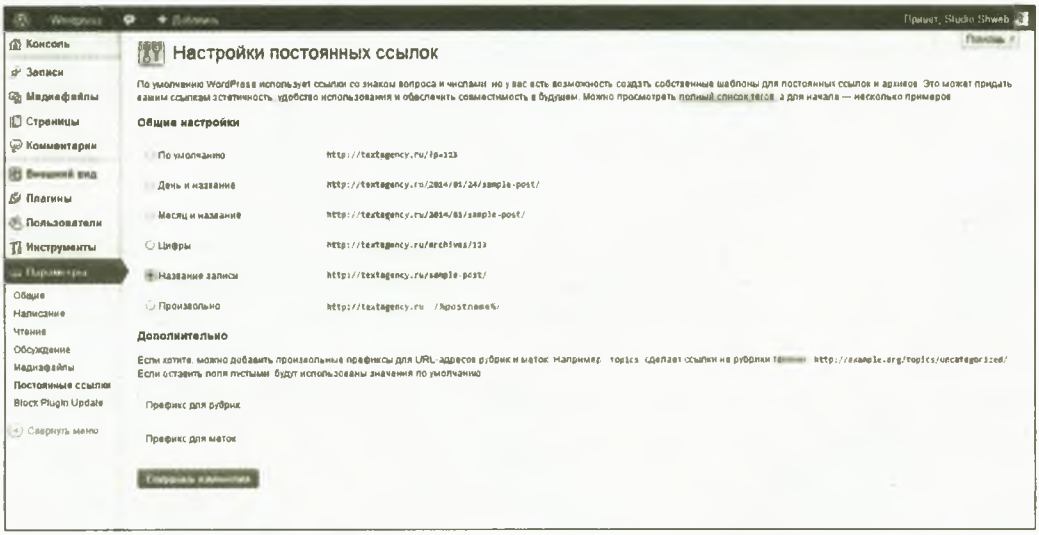


Рис. 12.3. Задание структуры постоянных ссылок в Консоли

Более короткие адреса в целом более выгодны, так как их проще печатать, однако все же необходимо поддерживать их внутренние дескриптивные свойства. Поэтому мы рекомендуем использовать функцию Post name (Имя записи), которая работает аналогично созданию произвольной структуры постоянных ссылок с помощью `/%postname%/`. Эта функция использует краткий заголовок вашей записи или страницы и создает симпатичный, подходящий для поисковых машин адрес, похожий на приведенный в примере выше.

В более ранних версиях WordPress обращение к этой функции каралось сложной и длительной обработкой во время поиска. Это было именно тем местом, где требовалась высокая эффективность в обработке запросов к базе данных. Однако в WordPress 3.3 правила отсылки запросов к базам данных были упрощены

и усовершенствованы, что привело к резкому повышению эффективности обработки во время поиска.

Помимо вышесказанного, в вашем распоряжении находятся два дополнительных поля на панели управления для перезаписи адресов рубрик и меток. Например, когда вы находитесь на странице рубрики, ее адрес обычно выглядит примерно так: http://example.com/category/cool-stuff/. Вы можете заменить слово category на любое другое по своему вкусу с помощью этих дополнительных полей. Вы сможете использовать в качестве рубрики просто букву *«р»*, а в качестве метки — букву *«м»*, что сделает ваш адрес более коротким, однако капелька творчества при использовании данных полей может привести к созданию весьма интересных структур адресов.

ПРИМЕЧАНИЕ

У Криса Шифлета в его блоге по PHP-безопасности есть интересная запись (http://shiflett.org/blog/2008/mar/urls-can-be-beautiful), в которой рассматривается то, как сделать адрес красивым. На момент создания записи Крис работал в OmniTI, и адреса на новом сайте содержали слова-действия с очень четким значением, например http://omniti.com/is/hiring и http://omniti.com/helps/national-geographic.

Дублирование контента

Если поисковой механизм просматривает ваш сайт и находит дублирующийся контент, а точнее, несколько путей, ведущих к одному и тому же содержимому, то он может разделить ваш рейтинг (и SEO-оценку) между этими повторяющимися страницами, что приведет к растворению вашего общего рейтинга в рейтингах отдельных элементов контента. Данный раздел посвящен тому, как избежать рассмотрения нескольких путей к одному содержимому в качестве отдельных элементов.

WordPress практически способствует дупликации контента. Ваши записи отображаются на главной странице и на странице категорий в каждой категории, к которой относится запись. Каждая метка создает страницу меток для соответствующего контента, кроме того, записи сохраняются в архивах определенного года и месяца. Все это не только создает несколько путей для получения доступа к вашему содержимому, что в предыдущем разделе считалось положительным моментом, но и тащит вас на дно в плане проблем с повторяющимся контентом. Дублирование содержимого вашего собственного сайта может как быть, так и не быть негативным моментом, решение по этому вопросу до сих пор не принято. Например, Google crawler (поисковый бот Google), который индексирует ваш сайт, имеет встроенный механизм для определения природы и причины дублирования контента. Этот поисковый бот знает, как обращаться с различными просмотрами одного и того же содержимого, такого, например, как версии для печати. Он также прикладывает все усилия для поддержания просмотров оригинального источника контента: например, для сайта на WordPress это шаблон single.php.

Кроме того, Google предлагает воспользоваться сайтом Webmaster Tools (Инструменты для веб-мастеров), где раскрываются секреты, как поисковый бот Google видит ваш сайт. Воспользуйтесь плагином Google XML Sitemaps Арне Брачхолда, доступным по адресу http://wordpress.org/extend/plugins/google-sitemap-generator/. Он создает XML-карту сайта, которую Google будет использовать при индексации, эта карта помогает поисковикам обнаружить все необходимое. Однако Webmaster Tools (Инструменты для веб-мастеров) содержит некоторое количество весьма интересных инструментов и свойств. Например, в разделе Optimization ▸ HTML Improvements (Оптимизация ▸ Усовершенствование HTML) вы можете увидеть дублирующийся контент, обнаруженный поисковым ботом, как показано на рис. 12.4.

Рис. 12.4. Google Webmaster Tools (Инструменты для веб-мастеров)

Дальнейший просмотр решений для повторяющегося контента позволит вам установить, какие именно страницы создают вам проблемы. Честно говоря, Bing.com от Microsoft содержит аналогичный набор инструментов такого же качества.

Кроме того, вам необходимо отредактировать файл `robots.txt`. Файл `robots.txt` содержит информацию для поисковых ботов о том, что индексировать не нужно. По умолчанию бот отчаянно индексирует все, что находит. Файл `robots.txt` сообщает боту о том, что индексировать некоторые вещи нельзя. Вы, конечно, полагаетесь на то, что бот будет играть по правилам, и вот с чего вам стоит начать в работе с файлом `robots.txt`:

```
Sitemap: http://www.example.com/sitemap.xml
User-agent: *
Disallow: /cgi-bin/
Disallow: /wp-admin/
Disallow: /wp-includes/
Disallow: /wp-content/plugins/
```

```
Disallow: /wp-content/cache/
Disallow: /wp-content/themes/
Disallow: /trackback/
Disallow: /feed/
Disallow: /comments/
Disallow: /category/*/*
Disallow: */trackback/
Disallow: */feed/
Disallow: */comments/
Disallow: /*?
Allow: /wp-content/uploads/
```

Если вы хотите включить сюда карту вашего сайта, не забудьте изменить адреса в файле `robots.txt` так, чтобы они совпадали с реальными адресами на сайте.

Обратные ссылки и отклики

Google повышает рейтинг вашей страницы, подсчитывая обращения к вашему сайту на основании *обратных ссылок*. Они представляют собой подтверждение вашего контента другими сайтами. Они появились как способность одного сайта уведомить своих читателей о том, что их может заинтересовать содержимое другого сайта, а также как сообщение другому сайту: «Эй, я рассказал о твоем контенте, вот ссылка». Обратные ссылки можно рассматривать как комментарии к вашему контенту, оставленные на удаленном сайте.

По умолчанию WordPress группирует комментарии и обратные ссылки вместе, в дальнейшем подтверждая, что некоторые комментарии являются удаленными, но для читателя такая схема может выглядеть очень запутанной. Обычно применяется отделение обратных ссылок от реальных комментариев в цепи комментариев. Тема Twenty Eleven делает это за вас с помощью шаблонов по умолчанию, использующих функцию `twenty_eleven()`, находящуюся в `functions.php`:

```
switch ( $comment->comment_type ):
 case 'pingback' :
 case 'trackback' :
 ?>
  <li class="post pingback">
  <p><?php _e( 'Pingback:', 'twentyeleven' ); ?> <?php comment_author_link(); ?>
  <?php edit_comment_link( __( 'Edit', 'twentyeleven' ),
  '<span class="edit-link">', '</span>' ); ?></p>
  <?php
  break;
 default :
 ?>
  <li <?php comment_class(); ?> id="li-comment-<?php comment_ID(); ?>">
  <article id="comment-<?php comment_ID(); ?>" class="comment">
```

При поступлении комментариев эта функция определяет, являются они реальными комментариями или обратными ссылками, и отображает их соответствующим образом. Вы можете посмотреть файл шаблона `functions.php` в Twenty Eleven, чтобы получить более подробное представление и проанализировать остальную часть

функционала. Вы также можете перехватить эту функцию с помощью дочерней темы и выстроить логику отображения по-своему. Эта функция позволяет провести четкую границу между активным обсуждением на вашем сайте, в котором могут участвовать посетители, и списком сторонних сайтов, на которых просто упоминается ваш контент. Граница может быть проведена как логически, так и визуально, что сделает ее более естественной для посетителя.

Отклики, в свою очередь, уведомляют сторонние сайты о публикации новой информации. В целом, ваш сайт на WordPress должен уведомлять службу обновлений, такую как Ping-o-Matic, при обновлении и появлении на сайте нового контента. Аналогично, если вы пишете о своем контенте на другом сайте на WordPress, ваш сайт может получать отклик от другого сайта и сообщать ему о появлении у вас нового контента. В этом отношении отклики похожи на обратные ссылки.

Служба рассылки откликов при обновлении является удобным способом привлечения трафика на свой сайт. Некоторые сайты собирают информацию, поступающую от служб обновления, и создают у себя информационные сайты со ссылками, полученными от этих служб. В теории случайные пользователи, просматривающие такие сайты, могут обнаружить ваш контент, касающийся интересующей их темы. В этом отношении отправка откликов очень напоминает работу RSS-кнопок или твиты, сообщающие о новом контенте.

Подписаться на использование службы уведомлений, такой как Ping-o-Matic, очень легко. Просто обратитесь по адресу http://pingomatic.com/, добавьте свой сайт, и служба начнет работать. Ничего сложного.

Метки и сайты с общим контентом

Technorati.com был одним из первых сайтов, внедривших подобную службу обновлений и создавших совокупность блогов. Метки Technorati позволяют вам сортировать ваш контент по категориям на www.Technorati.com. Проще говоря, вы добавляете на вашу страницу метку, которая ссылается на Technorati, и ваш контент группируется с похожими записями на основании метки. Функции меток Technorati, однако, бледнеют рядом с новыми процессами уведомления. Хотя сайт Technorati и не является важным дополнением к WordPress, каким был когда-то, он все же предоставляет удобную возможность для упоминания вашего контента без приложения особых усилий. Теперь у вас есть также множество способов для рекламы нового контента посредством социальных сетей, таких как Facebook, Reddit и Twitter. Например, в главе 11 рассматривается добавление кнопок со ссылкой на социальные сети в ваши записи, что является эффективным источником краудсорсинга для ваших методов уведомления через эти новые способы связи. Стимулируя ваших пользователей рекомендовать ваш контент и направлять его на сайты, содержащие метки, вы увеличиваете шансы на то, что ваш контент можно будет найти не только с использованием поисковых машин.

На практике функция отсылки запросов используется не очень широко. Чаще применяются различные приложения, обрабатывающие RSS-ленты различных

сайтов и записи или твиты, которые на них появляются. В некоторых ситуациях более эффективными оказываются уведомления.

Как веб-стандарты помогают обнаружить ваши данные

HTML — язык разметки текста в самом прямом смысле. Когда HTML еще только разрабатывался, предполагалось брать контент и размечать его осмысленным и разумным способом. Множество различных HTML-тегов помогали завершить работу. Все это предполагалось для чисто научного использования и контента, в основном имеющего аналогичную структуру.

Неожиданно на сцену вышли «акулы рынка» и приложили к делу свои грязные руки. Они жаждали красивых разметок, графики, объявлений о распродажах и очаровательных розовых пони. Ради достижения их целей дизайнеры и разработчики, как хорошие, так и плохие, отказались от исходной разметки и использовали все доступные инструменты для создания самых красивых сайтов в Сети. Они добавили табличную разметку.

В последние годы среди наиболее успешных разработчиков началось возвращение к оригинальной концепции. Возможно, вы из их числа, раз уж читаете эту книгу. Эти успешные разработчики осознали силу принципа разделения, такого как отделение презентации от контента, CSS от HTML. Они также осознали преимущества использования семантического HTML.

Семантический HTML

POSH представляет собой старый добрый семантический HTML (Plain old semantic HTML). За этой аббревиатурой скрывается концепция возвращения к истокам в отношении базового HTML для создания сайтов. Для придания лоска и блеска разработчики могут использовать CSS. Посмотрите, например, CSS Zen Garden...

Но почему надо использовать на вашем сайте POSH? На то есть несколько причин. Во-первых, это наилучший выбор для Сети будущего. В будущем, если вы продолжите использовать на своем сайте семантический HTML, производители браузеров продолжат его поддерживать.

Во-вторых, это облегчает валидацию и поддержку контента для разработчика. В документе на чисто семантическом HTML намного меньше лишнего, чем в документе со старомодным кодом. Сравните это:

```
<div style="
 background: #F0CCFA;
 border: 1px solid # D894EB;
 color: #f00;
 font-size: 2em;
```

```
 margin: .25em 0;
 padding: .5em;">
  This is my subheading
</div>
```

и это:

```
<h2>This is my subheading</h2>
```

А ведь это еще не старомодный код. В нем просто использована CSS вместо множества вложенных тегов `<font>` , которыми действительно перегружены старые HTML-документы. С надежным и валидным HTML намного проще работать. Действительно очень просто.

К слову, о простоте: очистка вашего HTML от хлама может ускорить загрузку страниц. Задумайтесь обо всей лишней разметке, перемещенной из области загрузки каждой страницы в сохраняемый в браузере CSS-файл. Это может существенно уменьшить вес ваших страниц.

Третьей причиной является доступность. Семантическая структуризация HTML повышает вероятность того, что программы чтения экрана смогут разобраться в вашем контенте и он будет иметь смысл для посетителя сайта.

Наконец, валидный семантический HTML способствует оптимизации для поисковых механизмов. Поисковые боты не очень умны. Им абсолютно все равно, насколько красив ваш сайт, им плевать на созданные вами новые графические решения. Их интересует исключительно контент. Да и решать за себя они не умеют.

Семантический HTML передает значение текста, который вы размечаете. Именно поэтому он и называется семантическим. Использование правильного HTML-тега для вашего содержимого — первый шаг вперед. Например, HTML позволяет вам использовать шесть уровней заголовков. Использование их в правильном порядке очень важно для оптимизации сайта для поисковых машин. Аналогично: использование иерархии сайта для создания его разметки также связано с демонстрацией поисковому боту порядка представления вашего содержимого.

Даже если вы сможете правильно вынести CSS в таблицу стилей, поисковый бот не сможет правильно определить значение данного HTML-кода:

```
<div class="pagetitle">My Site Is About Something Important</div>
```

Если вы используете тег `<h1>` , то бот узнает, что текст является заголовком целой страницы, связанной с контентом соответствующего объема.

```
<h1 class="pagetitle">My Site Is About Something Important</h1>
```

Пролистывая свое содержимое сверху вниз, вы должны использовать для дополнительного контента подходящие заголовки. Главным является то, что каждая страница вашего сайта должна содержать только один тег `<h1>` для индикации верхнего уровня отображаемой страницы. Вполне разумным будет зарезервировать `<h1>` для названия сайта, а для других ситуаций использовать заголовки

другого уровня. Единственной существующей проблемой является то, что название вашего сайта не меняется, соответственно, наиболее актуальным для каждой страницы будет не `<h1>`. Использование заголовков описанным способом делает `<h1>` абсолютно не важным; `<h2>` же будет содержать название отображаемой страницы и описывать ее содержимое. Обе стороны вопроса вполне ясны, а решение остается за вами.

Изображения всегда должны иметь атрибуты `alt`. Они информируют поисковый бот об отображении изображения, так как саму графику боты не видят. Этой информацией также пользуются программы чтения экрана.

Тег `<div>` подходит для блоков контента, а `<p>` — для параграфов. Используйте более значимые `<em>` и `<strong>` для выделения и придания особой выразительности вашему контенту. Далее в данной главе мы обсудим теги HTML5 для конкретных блоков контента, такие как `<header>`, `<footer>` и `<article>`.

Пользуйтесь подходящими списками для организации ваших данных. Упорядоченные списки (`<ol>`) и неупорядоченные списки (`<ul>`) дают прекрасную возможность передать информацию поисковому боту. Правильно отформатированные списки содержат семантически больше данных для оценки, чем параграфы, наполненные тегами `<br />`. Имеется также менее известный элемент — список с подзаголовками (`<dl>`), который очень эффективен при использовании парных списков информации, таких как «Часто задаваемые вопросы». Перечисление списков и описание их HTML-атрибутов может продолжаться бесконечно.

Говоря короче, семантический HTML напрямую состоит из подборки правильных тегов и использования их по назначению. Рекомендуем вам прочитать спецификацию W3C и изучить различные теги и их назначение. Добавление этих дополнительных инструментов в вашу мастерскую сделает вас как разработчика лучше, ваши страницы — легче, осмысленнее и доступнее, что принесет пользу вам, вашим посетителям и вашим рейтингам в поисковых системах.

Валидный HTML

Для вас как для разработчика проще поддерживать валидный HTML и валидный CSS. Это факт. Если ваш код правильно структурирован, вы можете модифицировать его быстрее. У нас до сих пор лежат древние разметки, полученные от наших заказчиков, никогда не пытавшихся обновить стиль своего сайта. Если бы вам довелось поработать хоть с одной из них в течение некоторого времени, вы бы с удивлением вспоминали, насколько это тяжело. Но когда-то это было все, что мы могли сделать.

Валидный HTML также помогает решать проблемы кроссбраузерного отображения. Все разработчики с ужасом ожидают момента, когда им придется тестировать их красивенький сайт в одном из более старых или нестандартных браузеров. И вы знаете, о каком браузере идет речь. Важно, чтобы разработчик выпил необходимое количество кофе и убрал все острые объекты, находящиеся на расстоянии вытянутой руки, перед тем как открыть окно браузера для тестирования. Что-нибудь

обязательно не заработает. Наличие валидного HTML является первым шагом в борьбе с дефектами отображения данного браузера. Всегда начинайте с чистого кода и уже потом принимайте меры для доведения отображения ваших страниц в таких браузерах до адекватного состояния. И верьте: однажды этот браузер может исчезнуть насовсем. К счастью, поддержка устаревших браузеров становится проще, так как пользователи начинают осознавать необходимость обновления или пробуют новые браузеры. Кроме того, за счет некоторых манипуляций с темами HTML дает вам возможность связывать отображение в конкретных браузерах с условными комментариями.

Для поисковой оптимизации это возвращение к проблеме «боты не очень умны». Валидный HTML делает ваш контент более удобным для восприятия бота и, соответственно, для присвоения рейтинга. Если ваш HTML недостаточно валиден, поисковый механизм может «потерять» контент, который для него невидим во время поиска завершающего тега или атрибута. Это может также серьезно ограничить контент, видимый для поискового бота, и негативно повлиять на способность вашего сайта получать высокие рейтинги. Браузеры более милосердны к невалидному HTML и пытаются отображать все, что могут, но поисковый бот работает на основании скорости и количества поглощенного контента. Бот просто пройдет мимо всего, что ему непонятно. Опять же, обратитесь к инструментам вроде Google Webmaster Tools (Инструменты для веб-мастеров Google) для того, чтобы увидеть, как браузер воспринимает ваш сайт.

Для валидации вашего HTML существует множество служб, включая собственную службу W3C, Markup Validation Service (Службу валидации разметки), которую можно найти по адресу http://validator.w3.org/. Кроме того, большинство инструментов разработки браузеров имеют плагины или расширения для использования службы валидации W3C.

Микроформат

Микроформат является усложненным родственником POSH. Главная идея заключается в добавлении к HTML простых тегов, присоединяющих контекстную информацию к HTML-контенту. Однажды вы увидите, как они работают, многие из них пользуются вполне естественными способами обращения с контентом, напоминающими встраивание XML в HTML. Целью микроформата является конвертация некоего содержимого в HTML так, чтобы оно оставалось валидным и было доступно инструментам для работы с микроформатом. Например, вы очень часто видите контактную информацию и адреса, написанные синтаксисом микроформата. Возможно, вы даже пользовались микроформатом раньше и просто не знали об этом.

Например, ранее упомянутые теги Technorati являются микроформатом. Атрибут `rel` якорной метки ссылается на Technorati.com и указывает на то, что страница, с которой вы переходите, была помечена для использования в Technorati. Вот пример микроформата:

```
<a href="http://technorati.com/tag/wordpress" rel="tag">WordPress</a>
```

Другим распространенным микроформатом, встроенным в WordPress, является XFN (Сообщество друзей XTHML). Этот микроформат представляет собой просто атрибут, который вы помещаете на ссылку для указания ваших отношений с человеком. Данная функция встроена в панель управления ссылками и также известна как blogroll (перечень ссылок на другие блоги).

Используя такую удобную панель управления, вы можете легко присваивать атрибуты микроформата вашим ссылкам, указывая, насколько и откуда вы знаете человека, на которого ссылаетесь. Например, рассмотрите настройки, приведенные на рис. 12.5.

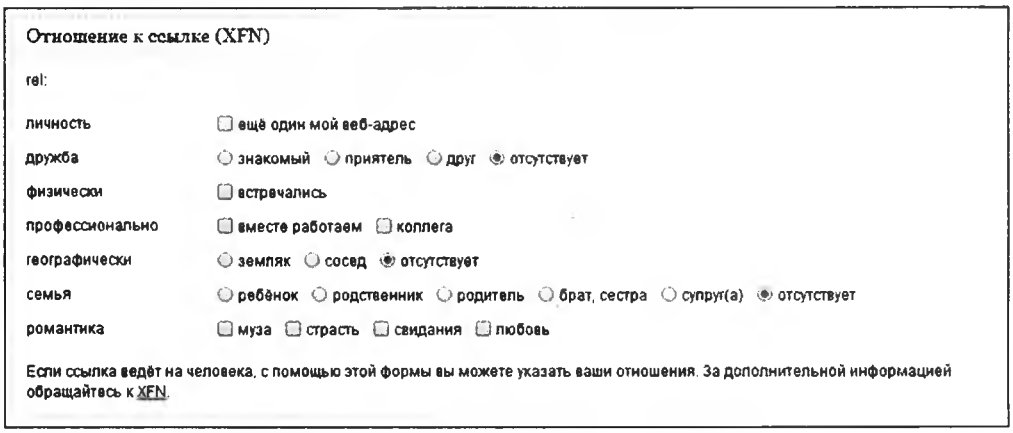


Рис. 12.5. Редактирование XFN-ссылки

Эти настройки будут выглядеть в HTML так:

```
<a title="WordPress.org" rel="friend colleague muse"
 href="http://WordPress.org">WordPress.org</a>
```

Это элементарный и в то же время действенный способ добавить важную информацию к ссылке. Ключевым моментом является простота метода. Для веб-браузера информация не влияет на отображение. На самом деле только недавно Internet Explorer хотя бы позволил разработчикам использовать атрибут `rel` в качестве селектора CSS .

Но только представьте себе, что будет, когда поисковые машины смогут создавать профиль в социальной сети на основании информации, содержащейся в микроформатах. Вы можете получить более подробное представление о XFN на сайте http://gmpg.org/xfn/.

Другим сильным микроформатом является hCard. hCard — это микроформат для отображения контактной информации человека или организации. Это простой, генерирующий визитные карточки формат HTML, используемый при отправке электронной почты и для ведения электронных адресных книжек в таких программах, как Microsoft Outlook и Mac OS X Address Book.

Вот пример hCard:

```
<div id="hcard-David-Damstra" class="vcard">
 <a class="url fn" href="http://mirmillo.com">David Damstra</a>
 <div class="org">Professional WordPress</div>
 <div class="adr">
  <div class="street-address">123 Main Street</div>
  <span class="locality">Grand Rapids</span>,
  <span class="region">MI</span>,
  <span class="postal-code">49525</span>
 </div>
</div>
```

Конечно же, данные были изменены, чтобы вы не смогли преследовать Дэвида. hCard — один из самых популярных микроформатов. Он очень похож на формат vCard, используемый программами электронной почты и электронными адресными книжками.

Отобразите приведенную выше hCard с помощью вашего сайта, и она будет выглядеть как безобидный блок адресов. Однако запуск этого же кода с помощью инструмента или бота, понимающего микроформат, может привести к намного более интеллектуальному примененению полученной информации.

Микроформаты позволяют внешним инструментам наилучшим образом использовать записи в вашем блоге, в идеале привлекая новых посетителей на ваш сайт. В свою очередь, они получают выгоду от внешних служб, пользуясь метаданными из ваших записей, имеющих теги в микроформате. Одним из примеров этого является плагин GeoMark, который конвертирует информацию о местоположении из вашей записи в теги микроформата GEO, хранящиеся как метаданные записи, а также передающиеся в RSS-ленту вашей записи.

В последнее время боты поисковых систем не оценивали данные в микроформате как-либо иначе, чем прочий контент вашего сайта. Однако микроформаты развиваются и продолжают набирать силу, соответственно, вскоре боты научатся распознавать их и собирать прилагаемые семантические данные. В заключение можно сказать, что микроформаты de facto становятся инструментом для разметки такого рода информации. Хотя микроформаты и рассматриваются ботами как традиционный контент, регулярное использование микроформата является работой, направленной на продвижение вашего контента в будущем.

Микроформаты — это инвестиции в будущее. Они дают относительно простую возможность структурировать специфический контент таким образом, чтобы в дальнейшем информация могла быть использована для чего-то познавательного или занимательного. Надеемся, что в будущем у вас появится возможность с помощью микроформата находить человека по имени и сразу же узнавать его социальный статус, или же искать предприятие и автоматически загружать его контактные данные в свой смартфон, или же искать конкретное место в конкретное время и получать список событий, происходящих поблизости. Объем информации возрастает, и инструменты для ее обработки не имеют права отставать.

HTML5

Пока мы говорили только о прогрессивных элементах HTML, давайте же теперь вкратце рассмотрим HTML5. Это информация, связанная не столько с WordPress, сколько с сетевыми разработками в целом.

Что такое HTML5? По большей части это следующее поколение стандарта HTML, который веб-разработчики используют уже долгие годы. Но само название слегка тяжеловато. Термин появился как рыночное определение, охватывающее многие и многие аспекты сетевых разработок и выходящее далеко за пределы базового синтаксиса HTML. Это совершенно необходимый термин, который должен быть на слуху у специалистов и определять самые передовые технологии в разработке, такие, например, какой была Web 2.0 в конце 2000 года. Когда вы слышите, как кто-то говорит об HTML5, вы должны спросить себя, имеют ли они в виду только HTML5 или же весь букет современных технологий разработки, включая HTML5, CSS3 и Javascript.

Давайте сосредоточимся именно на HTML5. Что входит в HTML5? На самом деле множество вещей, часть из которых вы можете использовать сразу же, часть стоит использовать выборочно, в зависимости от вашей целевой аудитории, и множество вещей, с использованием которых придется подождать до того, как браузеры начнут их поддерживать.

Самым важным свойством HTML5 и, скорее всего, свойством, о котором в первую очередь думают разработчики, являются новые теги. HTML5 содержит множество элементов HTML, которые более дескриптивны и имеют семантическую цель. При наличии небольшой помощи вы можете воспользоваться этими тегами уже сегодня и даже практически прямо сейчас. Если вы разрабатываете сайт или темы на WordPress уже в течение некоторого времени, вы легко узнаете привычные `<div id="header">` и `<div id="footer">`, присутствующие в абсолютном большинстве сайтов. В HTML5 эти элементы заменены на новые теги `<header>` и `<footer>`. Также созданы дополнительные теги, наилучшим образом подходящие для тем WordPress, включая `<nav>`, `<article>` и `<aside>`.

Как вы, вероятно, догадались, тег `<article>` непосредственно входит в парадигму записей и страниц WordPress. На самом деле уже тема Twenty Eleven использует теги для этого. Тема Twenty Eleven позволяет включить множество тегов HTML5 в шаблоны тем, а также является хорошей отправной точкой для освоения того, как использовать эти теги в ваших собственных индивидуализированных темах.

Другим хорошим источником для рассмотрения тегов HTML5 в действии является HTML5 Boilerplate Пола Айриша, который можно найти в Сети по адресу http://html5boilerplate.com/. Этот ресурс не предназначен конкретно для WordPress, но учитывает все наилучшие практики и рекомендации и связывает их в набор HTML5 для начинающих. Кроме того, существует несколько тем для WordPress, созданных с использованием в качестве источника HTML5 Boilerplate.

Быстрое предупреждающее замечание о тег-элементах HTML5: они не поддерживаются напрямую всеми браузерами, особенно браузерами Microsoft, предшественниками Internet Explorer 9. Тем не менее не все потеряно. Есть решения на JavaScript, заставляющие устаревшие браузеры распознавать новые теги HTML5 и стилизовать их соответствующим образом. Тема Twenty Eleven использует HTML5Shiv (http://code.google.com/p/html5shiv/) для получения доступа к этим функциям. Другой альтернативой, использованной HTML5 Boilerplate, является `modernizr.js` с http://modernizr.com. Modernizr включает HTML5Shiv, что позволяет устаревшим браузерам распознавать новые теги HTML5, а также содержит функцию распознавания дополнительных возможностей, позволяющую вам использовать дополнительные свойства HTML5 и атрибуты CSS3. Какое из двух решений вы выберете, зависит от ваших потребностей.

По нашему мнению, HTML5 вполне безопасен для использования на производстве при наличии подходящего JavaScript. Вы можете и должны разрабатывать новые темы с тегами HTML5. Сам факт того, что Twenty Eleven использует эти теги, является знаком одобрения со стороны Automattic. Просто проведите оценку браузера, который должен поддерживаться, и примите решение на основании ваших требований.

Помимо новых и более дескриптивных HTML-тегов, HTML5 включает широкий круг дополнительных свойств. Как уже упоминалось ранее, некоторые из них вы можете использовать сразу же, а с использованием других придется подождать до того, как браузеры начнут их поддерживать. Удобным сайтом для получения информации о том, что поддерживают браузеры, является http://caniuse.com.

Этот сайт отслеживает старые, современные и новые разрабатываемые версии популярных браузеров и оценивает степень поддержки ими новых свойств, включая HTML5, CSS3 и пр. На сайте не учитываются ситуации, когда вы прибегаете к помощи JavaScript, такой как использование HTML5Shiv или Modernizr.js. Сайт отслеживает только поддержку в исходной версии. Этот сайт может стать очень удобным ресурсом, если вы планируете свою работу и решаете, какой браузер вам поддерживать или какие свойства добавить.

Дополнительные свойства HTML5 включают автономное хранение, которое может пригодиться в сетевых приложениях для мобильных устройств и для ситуаций, когда у устройства отсутствует подключение к сети. Теги медиа и растровых изображений также доступны для передачи соответствующих файлов с информацией напрямую в браузер. Одним из особенно приятных свойств является наличие новых форм валидации и поддержки. Эти теги обновленной формы включают сам механизм валидации, встроенный в браузер, а также дополнительные возможности, такие как замещающий текст для форм и форматирования. В основном все новые свойства направлены на решение ваших повседневных проблем при работе с Сетью.

В HTML5 имеется множество свойств, узнать о которых вы можете здесь: http://www.html5rocks.com.

CSS3

Данный раздел посвящен каскадным таблицам стилей версии 3, или CSS3. Это новая итерация в создании стиля сайтов. На данный момент множество новых стилей CSS3 внедряется в браузеры за счет использования специфических префиксов. Это позволяет вам попробовать новые свойства в конкретных браузерах, но усложняет работу, если в вашем CSS-файле содержится 3–4 строки и вы хотите добиться одинакового эффекта в различных браузерах.

Опять же, http://caniuse.com является доступным ресурсом для определения того, какие свойства вам доступны во всех браузерах при использовании официального синтаксиса CSS3, а с какими вам понадобятся специфические префиксы. Еще одним аспектом является то, что современным разработчикам не нужно, чтобы все браузеры одинаково отображали сайт. Разумного сходства вполне достаточно. Помимо этого, добавление украшений и элементов дизайна в браузеры, которые их поддерживают, и отображение сайта в старых браузерах без этих украшений также является приемлемым. В целом это называется *прогрессивным улучшением*.

Типичным примером прогрессивного улучшения является округление углов. Представьте ваш дизайн-макет, на котором у отдельных элементов дизайна округлены углы. До появления CSS3 разработчикам приходилось прибегать к некоторым уловкам для получения округленных углов, включая изображения как в позитиве, так и в негативе, а также бесчисленные библиотеки JavaScript. С появлением CSS3 округленные углы стали требовать лишь применения стиля border-radius.

Вы можете применить этот стиль и проверить через свой браузер — отлично работает. Однако если вы возьмете border-radius с www.caniuse.com, вы обнаружите, что Internet Explorer 7 и 8 не поддерживают данный стиль, вот почему вы в первую очередь проверяете, работают ли ваши трюки. В данном случае вы можете обратиться к прогрессивному улучшению. Сайт все еще отлично отображается в Internet Explorer, однако некоторые элементы имеют квадратные углы. В браузерах, поддерживающих улучшение, сайт выглядит, как вы планировали. Тут стоит задуматься вот о чем: однажды (мы надеемся) все браузеры станут работать по одному стандартному принципу, и тогда (мы надеемся) ваш улучшенный дизайн сможет увидеть каждый. С другой стороны, пока вам еще приходится тестировать свой дизайн в более старых браузерах, если, конечно, у вас есть необходимость их поддерживать, чтобы убедиться, что сайт в них отображается адекватно.

Одним из самых мощных и практически сразу же доступных свойств CSS3 являются медиазапросы. Медиазапросы позволяют дизайнеру выстраивать сайт под конкретное разрешение экрана в ответ на новый опыт работы с сайтами для мобильных устройств. Эта тема раскрыта в данной главе немного более подробно.

HTML5 добавляет новые семантические теги, которые помогают сделать контент вашего сайта более понятным для устройств, отображающих или просматривающих ваши данные. Вкупе со стилями CSS3 это позволяет придать вашему содержимому новый стиль и добавить к нему новые атрибуты. Вместе эти свойства позволяют

упростить взаимодействие как с пользователями, так и с программами и устройствами.

Поиск по вашему сайту

Вы уже узнали, как сделать ваш сайт видимым и полезным для крупных поисковых машин за счет структуризации, организации и программирования с целью повышения рейтинга или, как минимум, получения справедливой оценки. Что же происходит, когда пользователь заходит на ваш сайт и пользуется встроенным поиском? Действуют ли те же правила и советы?

Ответ: и да, и нет. Принципы и практики SEO, которые мы обсудили, служат надежным фундаментом. Это проверенные правила, однако поисковые машины нередко задают свои или меняют их без предупреждения — добро пожаловать на Дикий Запад Интернета. Все перемены заложены во встроенном поиске WordPress.

Слабые стороны поиска по умолчанию

В исходной версии поиск WordPres, возможно, весьма хорош для небольших сайтов. В конце концов, именно с него WordPress начал эволюционировать, и «весьма хорош» — и вправду означает «весьма хорош». Однако исходный поиск WordPress имеет несколько серьезных недостатков, если применять его на сайтах большего размера. Кроме того, есть несколько сложностей, с которыми придется столкнуться.

Результаты сортируются по дате, а не по близости к тексту запроса. WordPress обожает выводить контент в хронологическом порядке. Хронология записей — сердце движка WordPress . Соответственно, поиск по умолчанию выдаст вам результаты по запросу в обратном хронологическом порядке. Поиск подвержен эффекту новизны.

Даже если у вас есть большая, прекрасно написанная статья на заданную тему, в случае существования более новых записей на ту же тему они окажутся выше вашей записи в результатах поиска. Соответствие тексту запроса роли не играет. Механизм поиска не умеет оценивать полученные результаты. Поиск просто просматривает все записи и весь контент страницы, и если ему попадается искомый термин, запись отправляется в список результатов поиска, которые потом сортируются и выводятся в соответствии с датой создания.

Это порождает следующий недостаток. Поиск просматривает только некоторую часть содержимого вашего сайта. Поиск по умолчанию обращается только к содержимому записи или содержимому страницы, но не к их заголовкам, комментариям, ссылкам, категориям, меткам или чему-то еще. Из предшествующего материала вы узнали, что заголовки имеют значение и посетители читают только заголовки, но если ваш интересный заголовок запомнится кому-то и он вернется на вашу страницу и начнет поиск по запросу, напоминающему тот заголовок, то этот человек может ничего не найти, так как заголовки не индексируются.

В идеале если ваш контент содержит интересные заголовки, то он может быть обнаружен с помощью поиска. Однако на вашем сайте намного больше контента, который может быть использован для усовершенствования поиска и повышения его эффективности или даже для расширения поиска. В конце концов, предметом интереса может оказаться один из комментариев.

В поиске отсутствует логика. Это значит, что в запросе вы не можете использовать, например, булев синтаксис. Поиск WordPress прямолинеен, это поиск по принципу «найди слово в записи». Булев синтаксис позволяет пользователям постоянно улучшать точность результатов, но это не случай WordPress. Поиск WordPress делает с ключевыми словами всякие глупости.

Например, попробуйте на сайте на WordPress использовать булев синтаксис с ключевыми словами в строке и поискать `keyword1 AND keyword2`. В этом случае вы хотите найти контент, содержащий оба ключевых слова. Вы обнаружите, что WordPress отнесся к `AND` как к еще одному ключевому слову и выдал вам контент, содержащий все три слова, то есть ключевые слова и слово `AND`. Вероятнее всего, нужных результатов вы не добьетесь.

Поиск WordPress таким же образом обращается с булевым `OR`. Попробуйте поискать контент по строке `keyword1 OR keyword2` или по строке `keyword1 OR keyword2`. Опять же, поиск WordPress обращается с `OR` как с остальными ключевыми словами. Теперь вам придется искать по каждому из ключевых слов и сравнивать результаты. В зависимости от типа вашего сайта вы заметите, что результаты поиска с `OR` не содержат такие же результаты, как общие результаты поиска по двум отдельным словам. Поэкспериментировав некоторое время, вы поймете, что поиск WordPress не умеет работать с обобщенными булевыми запросами.

Некоторые люди жалуются на то, что поиск WordPress не подсвечивает искомые слова в результатах. В некоторых ситуациях такое подсвечивание удобно, в других случаях оно не влияет на полезность результатов поиска. Скорее всего, личный выбор разработчика просто определяет полноценность этой функции в конкретном плагине. С другой стороны, эту проблему можно легко решить с помощью некоторого творчества, PHP и CSS.

Поиск WordPress достаточно хорош, но он не умеет пользоваться такими удобными инструментами, как поиск MySQL FullText, или посторонними поисковыми механизмами, такими как Lucene или Sphinx. Само собой, WordPress нуждается в том, чтобы процесс установки оставался простым, а зависимость от дополнительных пакетов программного обеспечения — минимальной. Дополнения определенно усложняют установку, но если разработчик хочет и может интегрировать дополнительные программные пакеты, то зачем мешать ему?

Некоторые из них кажутся весьма выгодными, какими и оказываются для определенных разработчиков. Однако поиск — весьма личная вещь. Разным разработчикам нравится встраивать разные функции и алгоритмы, и так как WordPress обладает разветвленной системой плагинов, то каждый разработчик имеет возможность получить желаемое. Вы можете как найти уже существующий

плагин, так и написать свой собственный, чтобы сгладить острые углы поискового механизма.

Альтернативные и полезные плагины

Совершенно очевидно, что мы не первыми обнаружили эти досадные недостатки в механизме поиска. Несколько весьма одаренных разработчиков задались целью создания плагинов, которые улучшают или замещают встроенный поиск. Существует великое множество плагинов, некоторые имеют конкретные цели, другие же работают со всей процедурой поиска. Далее приведены наиболее популярные поисковые плагины для WordPress.

Search Everything (http://wordpress.org/extend/plugins/search-everything/) Дэна Камерона из Sprout Venture позволяет несколько расширить поиск WordPress search, добавив индексирование различных источников контента, таких как комментарии, метки и категории. Search Everything также содержит настройку подсветки искомого термина, все настройки можно контролировать через панель управления администратора.

Relevanssi (http://wordpress.org/extend/plugins/relevanssi/) Микко Саари — еще один плагин, замещающий механизм поиска. Этот плагин существует как в бесплатной, так и в премиум-версии, которая включает поддержку дополнительных свойств. Он включает подсветку искомого слова, а также осуществляет поиск в дополнительных областях, таких как комментарии, метки и настраиваемые поля. Однако главным преимуществом данного плагина является то, что он использует синтаксис, знакомый пользователям Google. Данный плагин поддерживает операторы булевой логики AND и OR. Он также поддерживает поиск по фразам, заключенным в кавычки, и по частично совпадающим словам. Это несколько больше, чем обычный поиск, ожидаемый пользователем.

В качестве альтернативы вы можете получить представление об ожидаемом пользователями опыте с помощью Google Custom Search Engine (http://www.google.com/cse/). Google Custom Search Engine (CSE) — служба, предлагаемая Google и содержащая свой собственный набор существующих результатов поиска. Google предлагает для вашего Custom Search Engine две программы. Базовая бесплатная редакция позволяет настраивать внешний вид и параметры, но включает Google Ads. Вторая версия обходится в $100 в год и позволяет избавиться от рекламы и логотипа Google, если вам угодно. Обе программы интегрируются в ваш сайт на WordPress аналогичным образом.

Существует несколько плагинов для интеграции Google CSE в ваш сайт. В качестве альтернативы вы можете интегрировать его вручную с помощью шаблонов страниц и кода, предоставляемого Google. Шаблоны страниц подробно рассматриваются в главе 9, но ниже мы приводим краткий пример того, как ими можно воспользоваться.

Сначала войдите в панель управления Google Custom Search Engine и создайте новый пользовательский поисковый движок. После того как вы это сделаете, вы захотите модифицировать его стиль. Измените раскладку на двухстраничную. Это

позволит вам разместить окно поиска в любом месте сайта и при этом иметь отдельную страницу для вывода результатов. Сохраните настройки и получите код.

Теперь вам придется немножко поколдовать для того, чтобы создать страницу, необходимую для кода Google Custom Search Engine. Когда вы переключитесь обратно на свой сайт WordPress, вам понадобится отдельная страница для результатов поиска, и вы захотите создать для этой страницы конкретный шаблон. Он может быть таким же простым, как, например, этот:

```
<?php
/*
 * Template Name: Google Search Results
 */
get_header(); ?>
<div id="primary">
 <div id="content" role="main">
  <!-- google search engine results -->
 </div><!-- #content -->
</div><!-- #primary -->
<?php get_footer(); ?>
```

Обратите внимание на то, что в области контента до сих пор существует точка, куда вы вставите код, предоставленный Google. Сохраните шаблон файла и вставьте его в вашу тему для WordPress на сервере.

Вам необходимо заставить страницу, использующую вашу тему, выводить результаты. Создайте на сайте новую страницу. Так как вы используете шаблон предыдущей страницы, вам не нужно размещать ничего в области контента; ее заполнит за вас Google.

И наконец, вам нужно создать окно поиска в вашей версии WordPress. Вероятно, наиболее простым способом сделать это будет использование текстового виджета в одной из областей для виджетов. Просто скопируйте код окна поиска, предоставленный Google, и сохраните его как виджет.

Теперь вам нужно протестировать, как все это работает вместе. В зависимости от того, когда ваш сайт был вывешен, вам придется подождать некоторое время, пока Google индексирует ваши страницы.

Для того чтобы в полном объеме задействовать возможности Google Custom Search Engine, убедитесь в том, что вы подключили его к учетным записям Google Analytics и Google Webmaster Tools. Помимо этого, воспользуйтесь некоторыми свойствами Google Custom Search Engine, такими как Refinements (Улучшение) для изменения параметров и Promotions (Продвижение) для подсветки отдельных результатов поиска.

Доступ с мобильных устройств и адаптивный веб-дизайн

Доступ с мобильных устройств продолжает быть актуальной темой по причине широкого распространения на рынке смартфонов с быстрым Интернетом, а также

магазинов, которые процветают за счет продажи дорогостоящих приложений для таких телефонов. Forrester, исследовательская фирма мирового уровня, сообщает, что более половины населения планеты являются владельцами мобильных устройств (http://blogs.forrester.com/susan_huynh/12-02-21-mobile_internet_users_will_soon_surpass_pc_internet_users_globally). Google сообщает, что среди пользователей Google+ больше владельцев мобильных устройств, чем настольных компьютеров (http://www.engadget.com/2012/06/27/google-has-250-million-users-more-mobile-than-desktop/). Морган Стенли уже в течение нескольких лет утверждает, что в ближайшие годы доступ в Интернет с мобильных устройств превысит доступ с домашних компьютеров (http://mashable.com/2010/04/13/mobile-web-stats/).

Пользователи мобильных устройств явно являются той аудиторией, игнорировать которую вы не можете себе позволить. Вопрос состоит в том, сколько внимания стоит уделять этой растущей массе. Будете ли вы подстраивать пользовательский интерфейс под каждое устройство? Сможете ли вы успеть за ростом количества этих устройств? Ведь размер экрана и свойства API могут быть разными у каждого из них. Существует несколько взглядов на то, как с этим справляться.

Оставьте их в покое

Первой парадигмой, притом самой простой, является возможность оставить все как есть. Это означает, что браузеру мобильного устройства будет показан весь ваш сайт. Это позволит предоставить им в полном объеме все, что вы приготовили для пользователей домашних компьютеров.

Главная идея здесь в том, что более новые браузеры для смартфонов могут довольно неплохо отображать стандартные сайты. Знакомые с техникой пользователи этих устройств понимают, что возможности браузера ограничены, а экран невелик, и они не ожидают захватывающего пользовательского опыта. Короче говоря, решение остается за пользователем устройства. Планшеты и смартфоны позволяют приближать и пролистывать сайт, и пользователи любят эти функции. Если у вас есть маленькие дети, умеющие обращаться с планшетом, возможно, у вас появляются следы пальцев на экране домашнего компьютера, оставленные ребенком, который пытается пролистывать информацию и жалуется, что экран не работает.

На практике принцип невмешательства работает отлично. Вся информация, размещенная на сайте, остается доступной для мобильного устройства. Посетитель продолжает использовать знакомые приемы работы с сайтом и ведет себя как при работе с обычным браузером. Реальной проблемой является сочетание мелкого текста и маленького экрана.

Легкие версии для мобильных устройств

Другим подходом к проблеме является использование доступных технологий для создания отдельных тем для мобильных устройств, особенно хорошо это удается

с помощью мощностей WordPress. Темы для мобильных устройств обращают нас ко временам легких сайтов, соединения с Интернетом по телефонной линии и экономии ограниченной ширины полосы. Темы для мобильных устройств загружаются быстрее при использовании беспроводного соединения. Помимо этого, такие темы должны быть адаптированы для просмотра на маленьком экране и содержать в основном ту информацию, которую может искать пользователь мобильного устройства, — контактные данные или информацию о местоположении.

WPTouch — плагин, конвертирующий ваш сайт в приложение для iPhone. Плагин WPTouch был разработан Дейлом Магфортом и Дуэйном Стори из Brave New Code, он доступен по адресу http://wordpress.org/extend/plugins/wptouch/.

После установки этого плагина ваш сайт сможет автоматически обнаруживать браузеры мобильных устройств и предлагать им ваш сайт целиком, но через тему, специально предназначенную для мобильного устройства. Тема использует запросы AJAX и прочие эффекты, позволяющие создать иллюзию официального приложения. Кроме того, WPTouch содержит расширенную контрольную панель, позволяющую управлять всеми настройками.

WPTouch также содержит возможность создания настраиваемой главной страницы для браузеров мобильных устройств. Это фантастическое свойство позволяет разработчику получить настраиваемую главную страницу с минимальной информацией, реально необходимой пользователю мобильного устройства. Помимо подобной iPhone темы WPTouch содержит возможность ослаблять CSS и создавать тему, напоминающую привычную тему вашего сайта. WPTouch также предлагает пользователю возможность выбрать просмотр этой привычной темы для домашних компьютеров.

Предостережение: во-первых, если вы используете кэширующий плагин, описанный в следующей главе, вам придется отключить в нем отображение сохраненного контента при использовании браузеров для мобильных устройств, иначе кэширование перехватит функцию определения браузера WPTouch и будут отображаться традиционные темы. Во-вторых, каждая процедура определения мобильного браузера происходит немного по-своему. Будет ли считаться мобильным браузер iPad или же его экран достаточно велик, чтобы быть экраном настольного компьютера? Что делать Amazon Kindle, экран которого немного меньше? Стоит ли плагину вести себя по-разному в зависимости от наличия у устройства подключения по WiFi или через более медленную линию мобильной связи? Постарайтесь найти оптимальный компромисс между реакцией программного обеспечения и предпочтениями пользователя.

Адаптивный дизайн

Адаптивный дизайн — новомодный тренд. Впервые адаптивный веб-дизайн был предложен Этаном Маркоттом на страницах A List Apart в 2010 году (http://www.alistapart.com/articles/responsive-web-design). По сути, адаптивный веб-дизайн использует функцию медиазапросов CSS3, описанную ранее, для изменения разметки

и дизайна темы вашего сайта под размер экрана того устройства, на котором сайт просматривается. В теории это позволяет одновременно управлять и темой, и контентом и подгонять их под различные разрешения экрана. Существует отдельный набор контента для каждого размера экрана, который адаптируется или реагирует на среду просмотра через правила отбора CSS. На практике все еще более сложно.

Этот принцип работает с помощью медиазапросов. Они существовали до CSS3. Таблицы стилей для печати, являющиеся формой медиазапроса, используются разработчиками уже давно. Все, что изменилось в CSS3, — это то, что теперь медиазапросы поддерживают расчет разрешения экрана. Эти запросы позволяют использовать некоторые стили CSS, только если разрешение экрана находится в заданных вами пределах.

Особые хитрости адаптивного веб-дизайна не связаны с WordPress и выходят за пределы темы данной книги, однако давайте вкратце рассмотрим, как он может работать.

Простым способом превратить дизайн сайта на WordPress в адаптивный является изменение боковой панели. В браузере настольного компьютера вы видите все место на экране, и обычно боковая панель располагается слева или справа. Это традиционный сайт. Однако на экране мобильного устройства пространство существенно ограничено, а информация, находящаяся в боковой панели, может быть не так важна. Например, при просмотре на мобильном устройстве информация, находящаяся в боковой панели, сдвигается вниз относительно первостепенно значимой информации. Используя блок медиазапросов в вашей таблице стилей, вы можете изменить поток ваших блоков контента. Это сохранит вашему первостепенно значимому контенту роль более важной информации наверху и в центре небольшого экрана, переместив менее значимый контент на менее выгодную позицию.

Опять же, реальное внедрение адаптивного дизайна сайта заслуживает написания отдельной книги. В Сети доступно множество руководств, пояснений, удачных примеров и рекомендаций. Более того, уже существует множество адаптивных тем для WordPress, включая Twenty Eleven, которые вы можете попробовать, чтобы понять принцип их работы.

Адаптивный веб-дизайн в данный момент является популярным решением среди разработчиков, нацеленных на аудиторию пользователей мобильных устройств. Это решение, на котором построены многие сайты. Но чем более адаптивными становятся темы, тем более сложным становится рабочий процесс. Помнит ли кто-нибудь время, когда вы разрабатывали отдельные темы для Netscape Navigator и Internet Explorer? Уже скоро адаптивные темы вернут вас в те дни, и вы будете чувствовать себя так же, как когда переделывали бесконечное количество версий вашего сайта под различные размеры, притворяясь, что не делаете этого. Вы закончите свои дни, исправляя одну и ту же тему в разных разрешениях по сто раз. А так как мобильные устройства продолжают развиваться, ваша работа может стать еще более монотонной, если вы попытаетесь охватить все модели и разрешения. Обратной стороной этой медали является полный контроль, когда адаптивный дизайн является ровно настолько сложным, насколько вы ему позволите.

Темы для мобильных устройств продолжают быть перспективной областью сетевых разработок. Так как смартфоны с относительно рабочими браузерами становятся все более распространенными, наличие соответствующего функционала у любого сайта становится все более необходимым. Вы будете и дальше видеть темы, оптимизированные для мобильных устройств, и все чаще будут появляться адаптивные темы. Все больше тематических концепций и тем для начинающих будут включать адаптивные элементы, облегчающие весь рабочий процесс на начальном этапе.

Резюме

Мы рассмотрели вопросы взаимодействия с пользователем, обсудили, как сделать ваш сайт более интересным и доступным как для читателей, так и для устройств. Мы поговорили о технологиях HTML5 и CSS, а также обсудили основные принципы создания пользовательского интерфейса. Из следующей главы вы узнаете о производительности, масштабировании и безопасности вашего сайта, особенно в ситуации увеличения числа посетителей.

Статистика, масштабируемость, безопасность и спам

13

В этой главе:

- Добавление счетчиков трафика на веб-сайт
- Кэширование содержимого для увеличения информационной нагрузки
- Обеспечение жизнеспособности и безопасности сайта WordPress
- Предоставление разрешений пользователям

В предыдущих главах рассматривались способы эффективной и привлекательной подачи вашего исключительного контента, способы повышения вероятности того, что пользователи найдут именно ваш контент, а также способы объединения различных источников контента на вашем сайте WordPress. Что же происходит, когда все условия выполнены и множество посетителей стекается на ваш сайт? Теперь у вас есть действующий сайт, и перед вами встают новые задачи. В этой главе рассматриваются способы, которые позволят определить и оценить успешность вашего проекта; вы узнаете, как можно действовать в случае появления нежелательного содержимого и вредоносных посетителей, а также разберетесь, какие изменения могут потребоваться в связи с увеличением количества посещений.

Счетчики статистики

Просмотр статистических данных о трафике позволит вам оценить, какая именно информация, представленная на вашем сайте, больше всего привлекает посетителей. Благодаря этой информации вы сможете понять, какие данные используются, а какие — нет. Кроме того, статистика трафика предоставит вам возможность получить ценную информацию о посетителях сайта, об используемом ими оборудовании и настройках их программного обеспечения. Владея этой информацией, вы сможете соответственно адаптировать свой сайт, обеспечить поддержку браузеров, которые

используют ваши посетители, и тем самым дать им возможность получить больше удовольствия и пользы от посещения вашего сайта.

Статистические приложения предлагают ряд методов для сбора данных, и каждый из этих методов имеет свои преимущества и недостатки. Каждый пользователь выбирает свой метод для сбора статистических данных о трафике.

Собирать статистические данные о трафике можно несколькими способами. Один из давно применяемых способов — анализ лог-файлов. Если ваш веб-сервер настроен правильно, он создает лог-файл в случае каждого запроса и ошибки, которые обрабатывает. Некоторые приложения для сбора статистических данных могут анализировать такие файлы и переводить информацию в пригодные для чтения человеком формы.

Второй способ предусматривает размещение фрагмента кода, обычно JavaScript, на каждой странице сайта, что позволит связываться с центральным сервером, накапливающим информацию и переводящим ее в понятные человеку формы. Этот метод, в отличие от предыдущего, считается современным.

Для каждого из этих вариантов существует плагин WordPress. Кроме того, пакеты написаны с использованием разных языков. Следует определиться, какие значимые показатели вы можете получить, используя тот или иной пакет: например, определить количество посетителей относительно количества уникальных посетителей, количество посещений относительно просмотров конкретных страниц и просмотров конкретных страниц уникальными посетителями. Тип данных, получаемых при сборе статистики, зависит от ваших целей. Если ваша цель — увеличить количество посетителей и привлечь внимание к сайту при поиске через Google, через рекомендации в социальных сетях и других внешних агрегаторах, достаточно будет отслеживать количество пользователей, посетивших одну из страниц вашего сайта или потративших на посещение вашего сайта менее одной минуты. Если ваш сайт предназначен для общения и создания определенной среды, может оказаться более полезным оценить количество пользователей, посетивших ваш сайт повторно, или проконтролировать длительность пребывания отдельного пользователя на сайте (от момента входа на сайт до момента выхода), а также количество страниц, просмотренных каждым пользователем.

AWStats

AWStats — один из устаревших способов сбора статистических данных о сетевом трафике. На самом деле ранее применялся другой пакет программ, но он вызывал определенные проблемы в области обеспечения безопасности, и потому в качестве основного пакета для сбора статистических данных стал использоваться AWStats.

AWStats предлагает ряд счетчиков статистических данных для анализа лог-файлов. Такие счетчики можно запускать на сервере, или можно загружать лог-файлы на другое устройство и запускать их на нем, что имеет смысл при отсутствии прав доступа или разрешения на изменение конфигурации сервера, работающего с вашим

сайтом WordPress. Для использования AWStats используется язык Perl; этот пакет успешно применялся на серверах Apache и Microsoft IIS, однако для работы с IIS требовалась определенная корректировка форматов лог-файлов. Для установки и автоматического запуска AWStats на сайте вам необходимо ознакомиться с задачами по администрированию сервера. Будучи предназначенным для анализа лог-файлов, пакет AWStats может автоматически запускаться в фоновом режиме через определенную программу в системе Unix.

Поскольку пакет для анализа лог-файлов AWStats является серверным приложением, он с легкостью отслеживает поступающие на ваш сайт запросы. Вы можете расширить область собираемых данных, прописав в специальном теге JavaScript пакета AWStats требование к сбору дополнительной информации от браузера (например, о размере экрана) и к поддержке плагина браузера при использовании различных технологий.

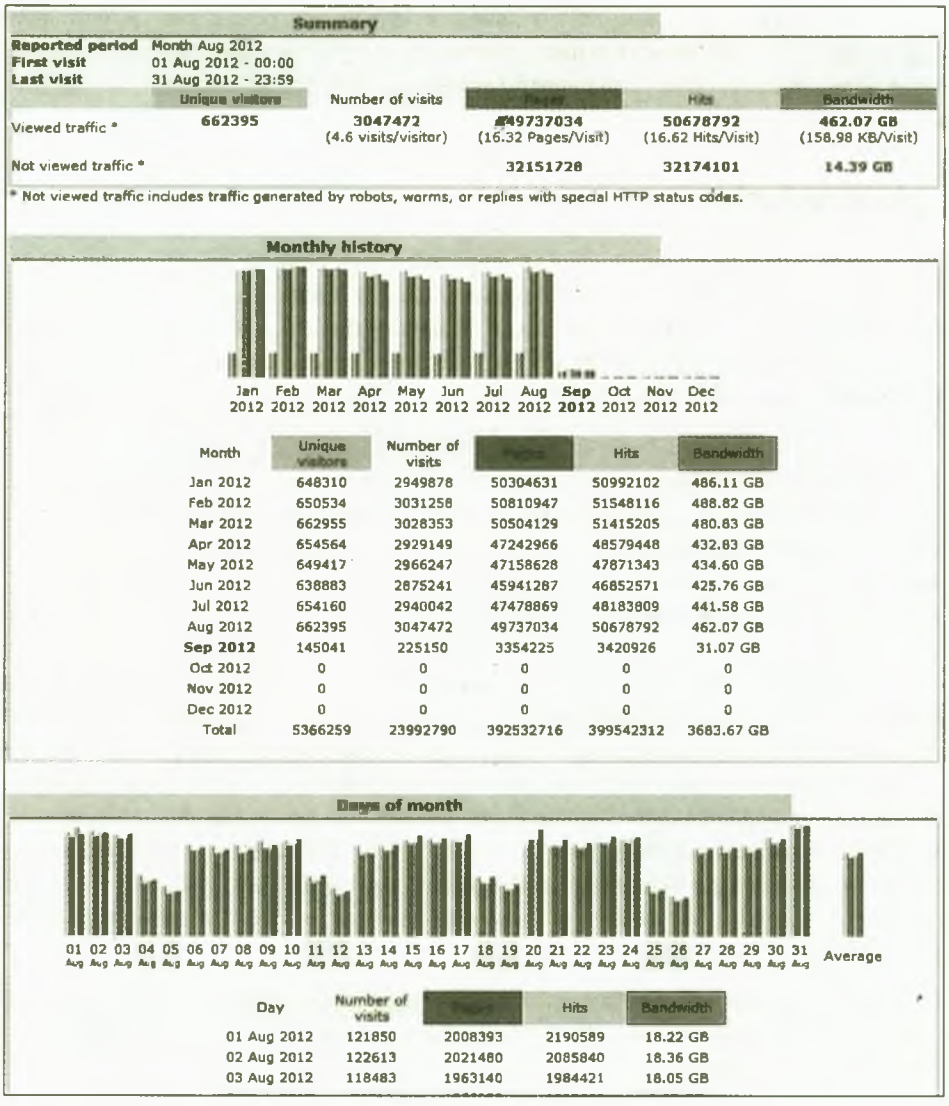
AWStats является одним из первых пакетов, разработанных для анализа статистических данных, с открытым исходным кодом. Этот пакет используется в течение достаточно долгого времени благодаря своей надежности, бесплатности и относительной простоте применения. В состав этого пакета входит ряд дополнительных скриптов и вспомогательных материалов, кроме того, этот пакет поддерживает различные источники, представленные в Интернете. Множество хостов используют пакет AWStats, и, как любой другой пакет программ с открытым исходным кодом, он обеспечен надежной поддержкой.

Одним из недостатков пакета AWStats являются периодические сбои при записи дат, которые в основном связаны со случайной ошибкой системного управления. Для предоставления определенных данных об истории и ускоренной обработки лог-файлов AWStats использует кэш-файлы.

В случае отображения дат в неправильном порядке или возникновения иных проблем с отображением времени такие кэш-файлы не позволяют анализировать какие-либо новые логи, что может повлечь за собой отображение противоречивых данных. Обычно достаточно вернуться назад и воссоздать все кэш-файлы. Можно найти определенные скрипты, которые помогут это сделать.

Другой проблемой, связанной с пакетом AWStats, является то, что браузеры в течение долгого времени не обновлялись, однако в настоящее время это уже не соответствует действительности.

Для пакета AWStats существует несколько различных плагинов WordPress, однако они не являются необходимыми для использования пакета AWStats на вашем сайте. Один из плагинов, предназначенных для использования с этим пакетом, обычно применяет данные, доступные для прочтения пользователем, и выкладывает их на интерфейсную часть вашего сайта. Также существует плагин, обеспечивающий вставку блока кода JavaScript, который выводит дополнительную информацию на каждую отображаемую страницу WordPress, благодаря чему пакет AWStats может собирать дополнительную информацию. На рис. 13.1 показан типовой отчет, созданный при помощи AWStats.



Summary

Reported period	Month Aug 2012				
First visit	01 Aug 2012 - 00:00				
Last visit	31 Aug 2012 - 23:59				
	Unique visitors	**Number of visits**	**Pages**	**Hits**	**Bandwidth**
Viewed traffic *	**662395**	**3047472** (4.6 visits/visitor)	**49737034** (16.32 Pages/Visit)	**50678792** (16.62 Hits/Visit)	**462.07 GB** (158.98 KB/Visit)
Not viewed traffic *			**32151728**	**32174101**	**14.39 GB**

* Not viewed traffic includes traffic generated by robots, worms, or replies with special HTTP status codes.

Monthly history

Jan 2012 Feb 2012 Mar 2012 Apr 2012 May 2012 Jun 2012 Jul 2012 Aug 2012 **Sep 2012** Oct 2012 Nov 2012 Dec 2012

Month	Unique visitors	Number of visits	Pages	Hits	Bandwidth
Jan 2012	648310	2949878	50304631	50992102	486.11 GB
Feb 2012	650534	3031258	50810947	51548116	488.82 GB
Mar 2012	662955	3028353	50504129	51415205	480.83 GB
Apr 2012	654564	2929149	47242966	48579448	432.83 GB
May 2012	649417	2966247	47158628	47871343	434.60 GB
Jun 2012	638883	2875241	45941287	46852571	425.76 GB
Jul 2012	654160	2940042	47478869	48183809	441.58 GB
Aug 2012	662395	3047472	49737034	50678792	462.07 GB
Sep 2012	145041	225150	3354225	3420926	31.07 GB
Oct 2012	0	0	0	0	0
Nov 2012	0	0	0	0	0
Dec 2012	0	0	0	0	0
Total	5366259	23992790	392532716	399542312	3683.67 GB

Days of month

01 Aug 02 Aug 03 Aug 04 Aug 05 Aug 06 Aug 07 Aug 08 Aug 09 Aug 10 Aug 11 Aug 12 Aug 13 Aug 14 Aug 15 Aug 16 Aug 17 Aug 18 Aug 19 Aug 20 Aug 21 Aug 22 Aug 23 Aug 24 Aug 25 Aug 26 Aug 27 Aug 28 Aug 29 Aug 30 Aug 31 Aug Average

Day	Number of visits	Pages	Hits	Bandwidth
01 Aug 2012	121850	2008393	2190589	18.22 GB
02 Aug 2012	122613	2021480	2085840	18.36 GB
03 Aug 2012	118483	1963140	1984421	18.05 GB

Рис. 13.1. Типовой отчет AWStats

Более подробная информация приведена на сайте в Интернете http://awstats.sourceforge.net/.

Пакет AWStats является испытанным и надежным способом сбора информации и подходит для использования с сайтами, которые должны оставаться автономными.

В отличие от примеров, рассмотренных в главе 11, которыми предусматривалось использование предоставляемого третьими сторонами контента для извлечения дополнительной прибыли за счет вашего сайта, этот пакет используется вашим веб-сервером и позволяет хранить все данные в одном месте. Если вы не хотите

учитывать статистические данные по вызовам JavaScript, ссылающимся на другие сайты, в связи с необходимостью поддержания работоспособности сайта или потому, что вы отказываетесь использовать JavaScript, AWStats — вполне подходящий для вас вариант.

Google Analytics

Сервис Google Analytics в настоящее время занимает важное место среди пакетов, предназначенных для сбора статистических данных. Сервис предлагает простой и, в общем, интуитивно понятный для пользователя интерфейс, который позволяет просматривать отчеты о статистике. Этот сервис работает за счет специального тега JavaScript на обрабатываемую страницу, который передает отчет о количестве посещений и информацию о браузере в Google для анализа.

И вот тут-то и обнаруживается камень преткновения, связанный с этим бесплатным пакетом для сбора данных, — вся информация о вашем трафике поступает в Google. В настоящее время многие люди используют сервисы Google для решения самых разнообразных задач, в том числе электронную почту, календарь и подсчет веб-статистики, однако некоторые люди по-прежнему не доверяют сервисам, подразумевающим передачу данных некому «Большому Брату», который следит за всеми. Несмотря на то что люди вроде бы доверяют сервисам Google, компания действительно может использовать получаемую информацию разными способами. Вы только задумайтесь, какое количество информации о работе браузеров и операционных систем получает Google; и все эти данные компания может использовать для целей службы контекстной рекламы AdSense и поиска по ключевым словам, полученным от вашего сайта. На сегодняшний день Google позволяет пользователям отслеживать случаи доступа к сайту за счет перекрестных ссылок AdWords и AdSense при помощи данных, предоставляемых сервисом Google Analytics.

Количество информации, связанной с использованием веб-сайтов и маркетинговых направлений, ошеломляет. Вам придется принять решение о сравнительном анализе выборок данных.

Сервис Google Analytics, вне всякого сомнения, ориентирован на маркетинг. Сервис предлагает множество встроенных мощных инструментов, освоение которых благоприятно скажется на качестве собираемых данных, и в том числе обеспечит лучшее сегментирование вашего трафика и индивидуальных отчетов о переданных данных. Например, в случае использования Google Analytics для отслеживания количества просмотров конкретных pdf-файлов из библиотеки вашего сайта, а также для email-маркетинга учитывается, какие компании обеспечивают наибольшее число переходов по ссылке. Для расширения области применения сервиса Google Analytics многие ресурсы представлены онлайн.

Например, ниже приведен фрагмент кода jQuery, обеспечивающий отслеживание ссылок на внешний документ при помощи сервиса Google Analytics. Этот код взят по ссылке http://css.dzone.com/news/update-trackingoutbound-click.

```
/* используйте jquery, чтобы отслеживать исходящие ссылки
* http://css.dzone.com/news/update-tracking-outbound-click
*/
$("a").click(function() {
var $a = $(this);
var href = $a.attr("href");
// проверяем, является ли ссылка внешней
if ( (href.match(/^http/)) && (! href.match(document.domain)) ) {
    // если так, то регистрируем событие
    var category = "outgoing"; // устанавливайте что хотите
    var event = "click";       // устанавливайте что хотите
    var label = href;          // устанавливайте что хотите
    _gaq.push(['_trackEvent'],category, event, href);
}
});
var fileTypes = ["doc","docx","xls","pdf","ppt","pptx", "rtf", "txt"];
$("a").click(function() {
    var $a = $(this);
    var href = $a.attr("href");
    var hrefArray = href.split(".");
    var extension = hrefArray[hrefArray.length - 1];
    if ($.inArray(extension,fileTypes) != -1) {
        _gaq.push(['_trackEvent'],"download", extension, href);
    }
});
```

Отслеживание внешнего трафика имеет смысл, если вы собираетесь использовать сайт в качестве справочно-информационного ресурса или хранилища специальных данных, так как в этом случае вам потребуется узнать, где посетители сайта ищут дополнительную информацию. Подробные сведения о типах документов, к которым ведут ссылки, помогут вам понять, какие виды контента наиболее востребованы. Естественно, вы сможете использовать дополнительные типы документов в случае, если вашей целевой аудиторией являются пользователи OpenOffice, PhotoShop и файлов в других форматах.

Для Google Analytics существует множество плагинов WordPress, что несомненно свидетельствует о популярности этого сервиса. Каждый из этих плагинов предлагает различные функциональные возможности, однако для использования любого из них требуется вставить фрагмент кода JavaScript на страницу сайта. Некоторые из плагинов предлагают дополнительные возможности для отслеживания внешних событий через панель наблюдения. На рис. 13.2 представлен скриншот стандартной панели наблюдения Google Analytics.

Более подробную информацию можно найти на сайте в Интернете http://google.com/analytics/.

С точки зрения практического применения Google Analytics представляет собой стандартный сервис для контроля сетевого трафика. В той или иной мере этот сервис используется практически всеми сайтами. Однако если вы не хотите ставить все на карту Google, возможны и другие варианты. Посетите сайт http://statcounter.com/, на котором представлен инструмент для анализа веб-трафика StatCounter, а также сайт http://haveamint.com/, на котором представлен инструмент Mint. В частности,

Рис. 13.2. Стандартный экран оповещений Google Analytics

Mint — самодостаточный инструмент, схожий с сервисом Google Analytics. Также существуют и новые возможности, например платформа Woopra (http://woopra.com/), которая в числе прочего собирает информацию о посетителях в реальном времени и периодически выводит эту информацию на экран.

Плагин WordPress JetPack

С нашей стороны было бы неправильным не упомянуть о плагине JetPack, созданном силами WordPress.com. По сути, этот замечательный плагин объединяет несколько функций, связанных с размещением информации, и может использоваться на самодостаточных сайтах WordPress. Этот плагин обладает множеством функций, в том числе обеспечивает интеграцию социальных сетей, расширение возможностей фотогалерей и, раз уж речь об этом плагине ведется в разделе, посвященном сбору статистических данных, само собой разумеется, что этот плагин позволяет собирать информацию о трафике.

Плагин JetPack схож с сервисом Google Analytics: для его использования фрагмент кода JavaScript вставляется в ваш шаблон HTML, что в дальнейшем позволяет

передавать отчеты на серверы WordPress.com для отслеживания трафика. Плагин JetPack был создан разработчиками для WordPress, поэтому статистические данные в контексте записей и страниц отображаются специфичным образом.

Кроме того, статистические данные, собранные плагином JetPack, отображаются непосредственно на консоли WordPress. Несмотря на существование плагинов, которые обеспечивают работу других пакетов по сбору статистических данных через iFrames, плагин JetPack по умолчанию использует вывод данных непосредственно на консоль. По сути, плагин JetPack изначально разрабатывался для встраивания в панель наблюдения. Некоторым конечным пользователям удобно управлять сайтом через консоль, как показано на рис. 13.3.

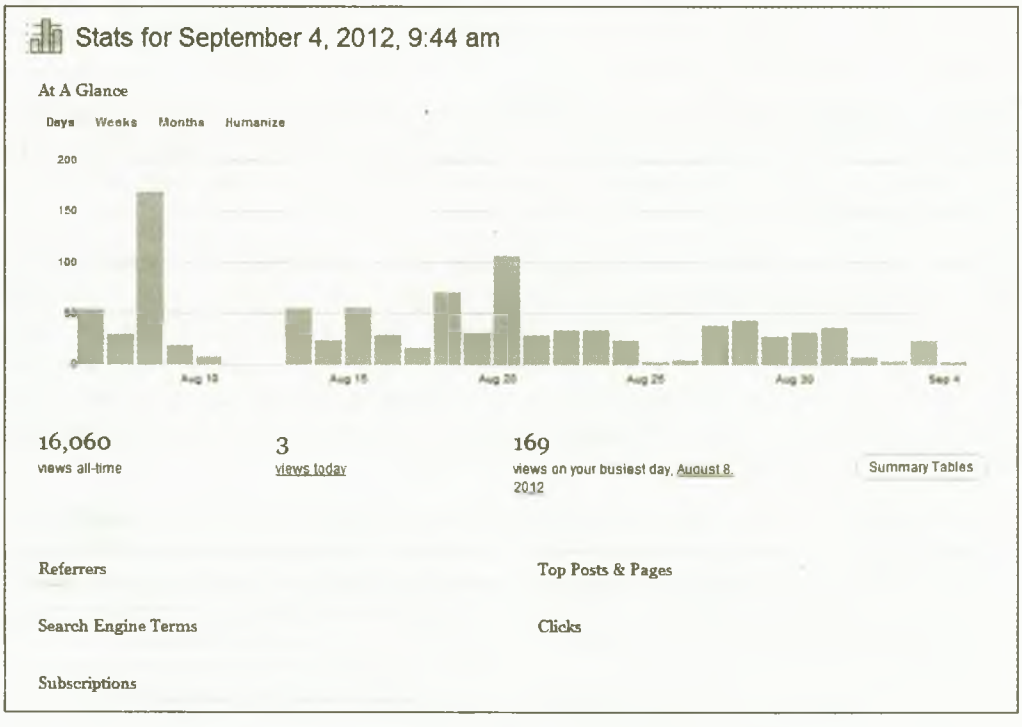


Рис. 13.3. Плагин JetPack, разработанный WordPress.com

Также плагин JetPack позволяет просматривать краткую информацию, полученную за последние 48 часов, которая выводится на панели администратора в верхней части экрана. Более подробная информация об этом приводится на сайте http://jetpack.me/.

В случае получения положительной статистики (когда ваш сайт набирает популярность, посетители активно участвуют в обсуждениях, а информационно-поисковые системы перенаправляют к вам на сайт новых пользователей) вы, вероятно, задумаетесь о масштабируемости своего сайта и начнете искать способы повышения общей эффективности системных компонентов WordPress.

Управление кэшем

WordPress представляет собой систему управления контентом, что по умолчанию определяет эту систему как средство для создания сайта с динамическим управлением. В современной технологической среде это означает, что обрабатываемое содержимое и все связанные с ним метаданные хранятся в базе данных. В случае поступления на сайт любого запроса происходит обращение к базе данных, что позволяет определить, какое именно содержимое следует отобразить, тогда как в случае статистического управления сайтом просто выбирается HTML-файл, расположенный в локальном каталоге на сервере в Интернете. Благодаря управлению содержимым посредством базы данных возрастает скорость отклика при попытке доступа к отдельной странице наряду с применением более мощных инструментов WordPress, обеспечивающих более длительное хранение, тщательный отбор и удобную организацию файлов.

И здесь начинают действовать основы информатики: вводя новый уровень абстракции, скорость которого ниже, чем скорость следующего уровня, вы тем самым запускаете механизм кэширования и одновременно увеличиваете быстродействие системы.

Имеет смысл рассматривать кэширование как последовательность методов доступа, начиная от пользователя и заканчивая базой данных MySQL. В каждой точке такой последовательности имеет место кэширование и настройка, однако, как и в любом другом случае, настройки быстродействия и показатели вашей системы зависят от способов доступа, типов содержимого и фактической рабочей нагрузки, действующей в определенной точке системы. Ниже приведен обзор иерархии кэширования WordPress с указанием проходимых сегментов:

- **Браузер.** В основном быстродействие браузеров ваших конечных пользователей зависит от оптимизированных CSS, графических характеристик и библиотек JavaScript. Поскольку эти параметры влияют на время загрузки отдельных страниц, мы рассмотрели их в главе 12, в разделе, посвященном взаимодействию с пользователем.

- **Веб-сервер.** WordPress и его плагины в основном написаны на PHP, интерпретируемом языке, но в целом выбор языка зависит от типа контейнера выполнения веб-сервера. Улучшение кэширования PHP на веб-сервере позволит частично ускорить прохождение пути от пользователя до базы данных на платформе WordPress.

- **Ядро WordPress.** Объекты кэширования, применяемые WordPress, фактически объединяются в кэш поиска по базе данных аналогично способу, применяемому сайтами с высокой масштабируемостью на основе MySQL, например такими, как Facebook. Изменение динамической генерации страницы на статический способ обработки (HTML) позволяет ускорить доступ к странице, однако иногда отмечается противоречивость при обновлении данных. Кроме того, как уже упоминалось в главе 11, для ускорения доступа можно кэшировать внешние и сложные информационные объекты, используя временные объекты.

- **MySQL.** Кэширование объектов на уровне базы данных предотвращает случайный доступ к дисковому запоминающему устройству, что приводит к преобразованию запроса в ссылку на ячейку памяти. Эта функция не зависит от включения плагина кэширования в память WordPress, но иногда является дополнительной.

Кроме того, фактическое преимущество каждого из описанных подходов зависит от ряда факторов, в том числе от уровня базы данных хостинг-провайдера, от того, можете ли вы самостоятельно настраивать веб-сервер и сервер баз данных, от частоты и сложности поступающих к базам данных запросов, а также от общей нагрузки на вашу систему WordPress.

Сложность системы WordPress

Во-первых, система WordPress достаточно сложна, и в этом ее преимущество. WordPress позволяет упростить процесс управления содержимым и за счет системы плагинов расширяет основные возможности, связанные с этим процессом. Однако предлагаемые дополнительные возможности и гибкость системы обеспечиваются за счет доступа к базе данных, обработки данных PHP, обработки уникальных требований каждого из плагинов и предпосылок каждой из специальных тем. Каждый плагин сопровождается непроизводительными затратами при отображении страниц, а качество кода конкретного плагина зависит от его автора. Используя анализатор выполняемых ветвей, например WebGrind или KCacheGrind, позволяющий вашему приложению выявлять возможные критические элементы кода, можно обеспечить графическое отображение сложности веб-приложения. Рассмотрим, к примеру, самый простой вариант установки WordPress с темой, предлагаемой по умолчанию. Запустив обычную загрузку страницы через этот профайлер и просматривая график, отображающий процесс выполнения этого запроса, вы увидите, что для отображения запрашиваемой страницы используется более 860 функций (рис. 13.4).

Рис. 13.4. Отображение сложности системы WordPress при помощи анализатора WebGrind

Можно оценить сложность системы WordPress даже без дополнительных средств, позволяющих в подробностях рассмотреть процесс ее работы. Ядро WordPress при помощи различных функций и действий собирает необходимые для вашего сайта данные, в том числе используя специальные дополнения для взаимодействия плагина и функциональные возможности темы.

Какой бы элемент вы ни добавляли на свой сайт, в том числе удивительную панель управления темой, которая позволяет задавать параметры прогона для вашего сайта, и плагины, которые анализируют ваш контент, связанный с определенными записями, на графике выполнения появляется все больше и больше блоков.

Не стоит паниковать по этому поводу и отключать плагины или останавливать свой выбор на самых простых темах. WordPress — уникальная система, и она может справиться с поставленными задачами. Гибкость, предлагаемая WordPress, а именно возможность изменения настроек во время работы, является мощной, но затратной особенностью. Альтернативой является статистически ограниченный набор функций и необходимость вставки новых фрагментов кода для применения новых функций, тогда как WordPress предлагает гибкую архитектуру плагинов. Каждый из предлагаемых плагинов и дополнительных параметров увеличивает функциональность вашей системы WordPress, поэтому вы и пользуетесь ими.

Просто признайте необходимость компромисса между функциональностью и эффективностью, где-то незначительного, где-то — более заметного.

На практике ваш сайт не так уж часто меняется. Вряд ли вы пытаетесь запустить на основе WordPress новую социальную сеть вроде Twitter (впрочем, используя тему p2, вы можете создать ее довольно точную копию), и отображаемое содержимое не изменяется каждые 30 секунд. В идеальной ситуации, когда содержимое на вашем сайте просматривается существенно чаще, чем он обновляется, что допускает возможность некоторой несогласованности при обновлении данных, вы сможете отслеживать слои кэширования от веб-сервера до MySQL.

Кэширование и оптимизация работы веб-сервера

Улучшение масштабируемости системы WordPress на уровне веб-сервера требует оптимизации выполнения PHP, а также изменения настроек веб-сервера. В обоих случаях вам потребуются полномочия администратора для доступа к файлам настройки веб-сервера.

Вам предстоит отслеживать запросы MySQL и кэширование объектов, однако независимо от конечного списка страниц WordPress за счет PHP обеспечивается отображение страницы и создание ее HTML-компонента. PHP представляет собой интерпретируемый язык, понятный компьютеру. Код каждый раз интерпретируется и компилируется в машинный код. У этого метода есть плюсы и минусы, но их обсуждение категорически не вписывается в формат этой книги.

Однако исполнительный уровень PHP может кэшироваться при помощи *кэша кода операции.*

Кэш кода операции PHP пытается замкнуть разрыв между интерпретированием работы системы и полностью компилированным кодом. APC (или альтернативный кэш PHP) представляет собой один из способов реализации, обеспечивающий кэширование и оптимизацию промежуточного кода PHP. Этот метод никак не относится к WordPress и применяется на подуровне PHP вашего сервера, и потому эта конфигурация не рассматривается в этой книге.

Для настройки APC вам необходимо разрешение на доступ к своему серверу. После настройки APC выполняется кэширование компилированных файлов PHP, находящихся в памяти WordPress. Основным недостатком является необходимость постоянной перезагрузки сервера при изменении страницы PHP. При изменении тем требуется перезапуск сервера каждый раз, когда в файл шаблона вносятся изменения. Более подробная информация об APC приведена на сайте http://us3.php.net/manual/en/book.apc.php. Еще раз напомним, что включение функции APC, а также включение/отключение веб-сервера возможно только при наличии у вас достаточных прав администратора.

Кэширование можно оптимизировать и далее — за счет кэш-памяти и размещения данных в сверхоперативной памяти, что также потребует наличия прав администратора для доступа к серверу. Кэш-память использует оперативную память вашего сервера для кэширования часто используемых объектов. Оперативная память обеспечивает бóльшую скорость, чем операции на основе файлов; поскольку процессы в сверхоперативной памяти используются как локальные присоединенные процедуры, они полностью протекают за пределами вашего веб-сервера. Более подробная информация о кэш-памяти приведена на сайтах http://us3.php.net/manual/en/book.memcache.php и http://memcached.org/. Основным преимуществом подобных систем оперативной памяти является добавление нового уровня кэширования на пути от WordPress до базы данных MySQL. Как уже упоминалось в главе 6, WordPress выполняет кэширование последних результатов, полученных в запросах SQL, созданных на основе базы данных MySQL, однако если ваш сайт последовательно загружает и затем сбрасывает содержимое кэша по мере того, как пользователи переходят от одного из популярных разделов сайта к другому, этот дополнительный уровень кэширования позволит повысить эффективность, так как будет сохранять в памяти результаты нескольких последних запросов. Кэш объектов WordPress сопоставляется с объектами, управляемыми кэш-памятью, что позволяет удовлетворять повторяющиеся запросы за счет содержимого кэша, а не за счет отправки в базу данных очередного запроса SQL. Такой уровень обеспечивает кэширование на уровне кода PHP, сокращая время выполнения PHP благодаря информации, помещенной в кэш ранее. Как будет показано далее, в потоке вычислений предусмотрены разнообразные возможности для применения кэширования.

При рассмотрении возможностей уровня PHP в контексте WordPress также следует оптимизировать (и обезопасить) конфигурацию вашего сервера `php.ini`. PHP пользуется таким успехом благодаря множеству встроенных возможностей, которые

при этом снижают уровень защищенности системы и облегчают разработчикам их задачу. Это всегда было и главным недостатком, и главным преимуществом PHP.

Откройте файл `php.ini` и отключите неиспользуемые расширения. Если они понадобятся в будущем, вы всегда сможете снова включить их.

Также воспользуйтесь этой возможностью для защиты своего контейнера выполнения PHP и ускорения его работы. Ниже приведены простые настройки. В зависимости от конфигурации вашего сайта может потребоваться отключение и других настроек для повышения безопасности и эффективности работы:

```
;Hide PHP for security
expose_php = Off
;Turn off for performance
register_globals = Off
register_long_arrays = Off
register_argc_argv = Off
magic_quotes_gpc = Off
magic_quotes_runtime = Off
magic_quotes_sybase = Off
```

Наконец, на уровне системного администрирования вы можете оптимизировать работу своего веб-сервера. В большинстве случаев конфигурация веб-сервера, предлагаемая по умолчанию, предназначена для обработки наиболее общих случаев использования. Разумеется, потребуется предпринять определенные действия, чтобы обеспечить соответствие определенным запросам. Так, например, сервер Apache идет в комплекте с множеством дополнительных модулей, которые позволяют работать с общими ситуациями. Если вам не нужны эти модули, отключите их. Это позволит сократить объем памяти, необходимый для работы Apache.

На практике можно добиться повышения эффективности работы Apache в условиях ограниченных ресурсов (например, при работе с частными виртуальными серверами с малым объемом памяти) за счет изменения конфигурации Apache PreFork. Конфигурация по умолчанию требует достаточно большого объема памяти, но вы можете ограничить требования к объему памяти в зависимости от нагрузки на ваш сайт и конфигурации вашей системы. Например, в случае размещения сайта с низким уровнем посещения на общем сервере с малым объемом памяти можно изменить файл конфигурации Apache2 следующим образом (при условии, что вы используете Apache2):

```
<IfModule mpm_prefork_module>
StartServers 3
MinSpareServers 3
MaxSpareServers 3
ServerLimit 50
MaxClients 50
MaxRequestsPerChild 1000
</IfModule>
```

Приведенные настройки предназначены для сайта с относительно низким уровнем посещения, расположенным на сервере с малым объемом памяти. Сложно

однозначно оценить уровень эффективности, которого можно достичь таким образом, однако внесенные изменения удивительным образом повлияют на время отклика веб-сервера на запрос от вашей системы WordPress.

Разумеется, эти настройки следует скорректировать в зависимости от ваших требований.

Учтите, что набор серверного программного обеспечения LAMP приобрел такую популярность только потому, что в большинстве случаев его можно использовать непосредственно после установки. Это связано с тем, что конфигурация этого набора по умолчанию предназначена для общих случаев применения и потому может использоваться в самых разных ситуациях. Впрочем, со временем может потребоваться изменение конфигурации в зависимости от конкретной ситуации.

Возможность настройки отдельных компонентов набора LAMP говорит сама за себя. Как и система WordPress, набор LAMP исключительно популярен благодаря своей гибкости и возможности одновременного решения разнообразных задач. Управление и администрирование компонентов LAMP — необходимые навыки для разработчика полноценных решений. Если вы потратите какое-то время на изучение предлагаемых инструментов и научитесь эффективно их применять, вы не пожалеете об этом.

Кэширование объектов WordPress

Целью кэширования на веб-сервере является сохранение часто вызываемых файлов и распространенных фрагментов кода в памяти для облегчения их использования. В контексте WordPress кэширование обеспечивает передачу запроса страницы без обращения к дополнительному фрагменту кода или без доступа к базе данных, что, по сути, сводится к замыканию PHP-ядра WordPress и обеспечивает непосредственное статическое отображение вашей страницы.

Кэширование объектов обеспечивает сохранение ряда часто используемых и ценных блоков данных в памяти. Эта процедура схожа с процедурой переходного кэширования, которая была описана в главе 11, однако переходные значения вы как разработчик задаете самостоятельно, тогда как процедура кэширования объектов определяется ядром WordPress. Гибкость процедуры кэширования объектов позволяет исключить влияние изменения части информации на все содержимое кэша, так как изменяются только сохраненные в кэше объекты, претерпевающие изменения. Однако кэширование объектов по-прежнему требует от плагина выполнить PHP-код, чтобы определить, какие из фрагментов кэша все еще актуальны; от WordPress также требуется выполнить PHP-код для сведения частей страницы вместе и ее последующего отображения. Как уже упоминалось ранее, оптимизация PHP-среды вашего веб-сервера и включение кэширования объектов на уровне WordPress, как результат, являются псевдонезависимыми.

Иногда предлагаемое вами содержимое достаточно статично или обращение к нему происходит так часто, что необходимо замкнуть блок кода PHP и кэш

объекта, например, в случае, когда ваша страница отображается в топе новостных сайтов Reddit и Slashdot. Например, это можно сделать, преобразовав соответствующую страницу в статический HTML, и обрабатывать ее напрямую веб-сервером. В конце концов, это именно то, для чего изначально разрабатывался веб-сервер.

Мы считаем, что наиболее подходящим для этой цели плагином является WP-Super Cache, разработанный Доннча Окайом. WP-Super Cache создан на основе плагина WP-Cache и является его улучшением. (Как вы можете заметить, это СУПЕРплагин.) WP-Super Cache функционирует в различных режимах, в том числе в режиме перезаписи статических файлов HTML. Однако для работы со статичными файлами HTML-плагину требуется `mod_rewrite`, и поэтому этот плагин работает только с серверами Apache.

Для WP-Super Cache предусмотрена расширенная панель управления, которая позволяет администратору сайта корректировать настройки в зависимости от конкретных целей. Плагином предусмотрен даже режим полной блокировки на случай интенсивного трафика.

Могут применяться и другие плагины для кэширования, например W3 Total Cache. Многие плагины для кэширования основаны на базовой теме, создающей фрагменты HTML-кода для страницы и обеспечивающей ее кэширование при помощи URL, используемого в качестве ключа для доступа к этой странице. Каждый плагин предусматривает различные способы определения недействительности содержимого кэша и срока действия страниц, и все они нарушают общую динамическую природу генерации страниц в системе WordPress. Если вы собираетесь изменить свою тему, добавить новые плагины или каким-то иным образом изменить поток данных, передаваемых от MySQL к браузеру пользователя, отключите кэширование WordPress и проверьте, работает ли система с внесенными изменениями, либо часто очищайте кэш, чтобы при проверке эффективности с учетом внесенных изменений видеть заново сгенерированные страницы.

Временный кэш

В главе 11 рассматривался временный кэш, однако в этой главе мы снова возвращаемся к этому аспекту. Временный кэш представляет собой кэш, определяемый разработчиком, который вы используете в собственном коде. Такой кэш применяется для хранения данных, полученных локальной версией вашего сайта удаленным способом для повторного использования. Именно этот способ применения в основном и рассматривался в главе 11 в связи с временным кэшем.

Другим способом применения временного кэша является хранение сложных расчетных данных. Например, в вашем шаблоне `functions.php` может присутствовать функция, направляющая сложный запрос через вашу сеть WordPress, включающую в себя несколько сайтов, с целью создания средства общесетевой навигации. Эти данные обычно нечасто претерпевают изменения, однако обработка запроса достаточно важна и может привести к замедлению отображения страницы.

При помощи временного кэша вы можете обрабатывать сложные запросы и сохранять их во временном кэше, чтобы следующему посетителю не приходилось ожидать выполнения запроса SQL. Использование временного кэша в этом качестве повышает работоспособность сайта за счет уменьшения количества запросов, поступающих в базу данных.

Кэш запросов MySQL

Выполняя исследовательскую работу при написании этой книги, мы установили систему WordPress в предлагаемой по умолчанию конфигурации, использовав тему «Twenty Eleven». Дополнительные плагины не устанавливались, то есть была установлена исключительно «система из коробки». Для обработки индексной страницы WordPress использует более 20 запросов MySQL. Убедитесь в этом сами, просто добавив приведенные ниже строки кода PHP в файл шаблона `footer.php`:

```
<?php echo get_num_queries(); ?> queries.
<?php timer_stop(1); ?> seconds.
```

Перезагрузите страницу, и в ее нижней части отобразится количество запросов и время, затраченное на их обработку.

От вас требуется оценить свой сайт, но, по всей вероятности, содержимое, сохраненное в базе данных, не может изменяться достаточно быстро, чтобы обеспечить загрузку всех обращений к базе данных по каждой странице. Перевод URL в запрос MySQL рассматривался в главе 5, а в главе 6 рассматривались исходные модели данных, так что объем трафика, необходимого для отображения простой индексной страницы, не должен вас удивить.

Кэширование WordPress обеспечивает ускорение доступа к содержимому, получаемому от MySQL. Если вы хотите дополнительно повысить эффективность MySQL и обеспечить более высокую надежность отклика на запросы от ядра WordPress, потребуется изучить данные о кэше запросов MySQL. Кэш запросов MySQL хранит результаты задачи `select (выбрать)` на случай поступления идентичного запроса, так как в этом случае из оперативной памяти можно мгновенно извлечь ранее полученные результаты. Это позволит существенно увеличить время отклика на запрос при условии, что данные претерпевают не очень серьезные изменения; в случае более серьезного изменения данных обновленные данные не могут отображаться мгновенно.

Для того чтобы включить кэш запросов MySQL, вам потребуется изменить файл конфигурации MySQL на сервере, однако для того, чтобы это сделать, понадобится разрешение компании — поставщика услуг хостинга. При изменении файла конфигурации MySQL вы можете увеличить предел памяти, например:

```
# enable 16 MB cache
query_cache_size   = 16M
```

В этом случае важно не устанавливать слишком большие значения. Выделение слишком большого объема оперативной памяти для кэширования запросов MySQL

может отрицательно сказаться на других подсистемах вашего сервера. Во всем нужно соблюдать равновесие. Простое включение кэширования таких запросов приводит к возникновению непроизводительных затрат при управлении процессами MySQL, но тем не менее вы в итоге оказываетесь в выигрыше.

Выравнивание нагрузки на ваш сайт WordPress

В какой-то момент, как мы надеемся, вы достигнете предела работоспособности пакета программного обеспечения, установленного на физическом сервере. В этом случае потребуется выровнять нагрузку на ваш сайт WordPress, задействовав один или несколько дополнительных серверов. Серверы можно добавить для расширения возможностей сайта в области обработки запросов, или же они могут использоваться в качестве ресурса, используемого при сбое, что повысит доступность вашего сайта. Независимо от выбранного варианта выравнивание нагрузки на ваш сайт, по сути, решит обе задачи, однако этот вопрос сам по себе достаточно сложный. Ниже вкратце рассмотрены сложности, с которыми вы можете столкнуться, пытаясь выровнять нагрузку на динамически отображаемый сайт.

В первую очередь вам потребуются средства для выравнивания нагрузки. Применение обычного подхода — использования цикличного DNS для передачи последовательных запросов HTTP между серверами — может повлечь за собой определенные проблемы, особенно в связи с cookie-файлами сеанса. Вам понадобится легитимное средство для выравнивания нагрузки, чтобы справиться с этой задачей. В качестве средства для выравнивания нагрузки может использоваться такой пакет программного обеспечения, как Pound (http://www.apsis.ch/pound/), или такое полноценное аппаратное решение, как F5 BIG-IP (http://www.f5.com/products/big-ip/). Оба варианта решения справятся с «липкостью» сессии и выравниваем нагрузки на ваш сайт.

Второй проблемой может стать синхронизация динамических данных, передаваемых между вашими двумя (или более) сайтами. Допустим, администратор вашего сайта успешно вошел в систему на одном из сайтов, разместил новое содержимое и загрузил на сайт новые изображения, но последующий запрос (при выравнивании нагрузки на сайт) может быть перенаправлен на другой сервер, где новое содержимое еще просто не существует.

Для начала обратите внимание на директорию «Загрузки». Содержимое загружается через консоль WordPress в директорию «Загрузки» установленной системы WordPress. По умолчанию содержимое загружается в `/wp-content/uploads/`. Однако вы можете изменить директорию «Загрузки» через меню Settings (Настройки) ▸ Media Dashboard (Панель наблюдения за материалами). В зависимости от размещения папки «Загрузки» вы сможете использовать более короткие адреса URL.

На этом этапе у вас появляются варианты, из которых можно выбирать. Один из вариантов — использование общей папки, к которой получат доступ оба сетевых

сервера. Вероятнее всего, это будет совместно используемый ресурс NFS/Samba, расположенный на третьем сервере, который вы будете использовать в качестве сервера MySQL. Второй вариант — использование программы rsync или аналогичного инструмента для координирования загрузок на двух серверах и контроля размещения основных фондов. Использование общей папки позволяет предоставлять доступ к основным фондам сразу, однако в связи с этим решением возникает точка отказа. Использование программы rsync позволяет синхронизировать (дублировать) данные и исключить точку отказа, однако влечет за собой задержку появления нового содержимого на отдаленных терминалах. Вам предстоит выбрать наиболее подходящее решение в зависимости от конкретных требований.

Также сложности могут возникнуть в связи с динамическими данными, сохраненными в базе данных. Допустим, ваша база данных не имеет слабых мест и потребность в выравнивании нагрузки на сайт не связана с ней. В этом случае можно использовать третий сервер в качестве сервера для размещения базы данных. Оба веб-сервера могут считывать данные из одного источника. Этот вариант может предложить более надежную архитектуру развертывания, если доступ к серверу, на котором размещается ваша база данных, не обеспечивается непосредственно через Интернет на общих основаниях, однако также это влечет за собой возникновение точки отказа. В этом случае технически выравнивание нагрузки выполняется только на конечных серверах.

Добавление второго сервера для размещения базы данных обеспечивает дублирование системы, но влечет за собой сложности в связи с синхронизацией двух таблиц базы данных MySQL. Серверы MySQL можно настроить на дублирование данных в режиме «ведущий-ведомый». Технически это не обеспечивает выравнивания нагрузки, поскольку в единицу времени предоставляется доступ только к одному серверу, однако такой тип конфигурации обеспечивает дополнительное дублирование данных. Изменения, внесенные в ведущую базу данных MySQL, дублируются в ведомой базе данных практически в режиме реального времени через протокол журналирования. В случае сбоя работы ведущей базы данных в ведомой базе данных информация сохраняется в полном объеме, и оттуда ее можно списать вручную.

И наконец, существует специальное решение WordPress для случаев использования нескольких серверов с дублируемой базой данных. HyperDB (http://codex.wordpress.org/HyperDB) был разработан силами Automattic для того, чтобы обеспечивать соответствие требованиям WordPress.com к объему передаваемой информации. HyberDB полностью заменяет модуль доступа к базе данных, встроенный в WordPress, и обеспечивает возможность использования нескольких баз данных, сегментирования вашей базы данных или ее разбивки на разделы при использовании нескольких серверов, а также позволяет дублировать данные и обеспечивает многоуровневый автоматический ввод резерва. Однако, к сожалению, документация на этот пакет далека от завершения.

Как вы уже, наверное, поняли, выравнивание нагрузки на сайт для повышения его эффективности и доступности является исключительно непростой задачей.

Существует множество систем, которые могут обеспечить соответствие широкому спектру требований. В кратком обзоре, приведенном в настоящем разделе, опущено множество нюансов и сложностей, с которыми можно столкнуться при развертывании WordPress в среде, обеспечивающей высокий уровень доступа. Сети «облачных» вычислений и доставки содержимого в настоящее время являются очень актуальными, и по мере развития этих услуг и технологий они, несомненно, будут все чаще использоваться WordPress для решения сложных задач и резервирования системы.

Работа со спамом

По мере развития вашего сайта, возрастания его популярности и увеличения трафика он начинает привлекать спамеров. Если вы обнаруживаете на своем сайте неожиданные записи или видите через Консоль учетные записи пользователей, которых вы не создавали, значит, безопасность вашего сайта под угрозой. Эти проблемы будут рассмотрены далее в настоящем разделе. Вероятнее всего, в таких записях будут содержаться различные нежелательные комментарии (спам), что является побочным эффектом популярности.

Обычно нежелательные комментарии, или же спам, содержат список ссылок или же не имеют смысловой нагрузки: в них сообщается, что автору понравился ваш стиль письма, и прилагается ссылка URL для перехода на какой-нибудь низкопробный ресурс. В любом случае целью спамера является написание нежелательных комментариев для создания большого объема сетевого содержимого, которое будет отсылать пользователей к сайту спамера. При этом спамер пользуется алгоритмами повышения популярности страницы, применяемыми Google, и аналогичными сервисами, которые придают вес входящим ссылкам. Наилучшим способом борьбы со спамом является его удаление, благодаря чему спамеры лишаются возможности использовать ваш сайт в качестве площадки для увеличения собственной популярности.

Существует три основных подхода к решению этой проблемы: запретить всем пользователям оставлять комментарии, затруднить спамерам доступ к комментированию на вашем сайте и использовать автоматическое обнаружение спама. Естественно, отключение возможности комментирования (через Консоль) — это скорее крайняя мера, которая не соответствует основной задаче вашего сайта, а именно взаимодействию с читателями. С другой стороны, если вы решитесь на такую меру, помните, что изменение настроек размещения записей через панель управления повлияет только на будущие записи; вся информация, ранее размещенная в вашем блоге, по-прежнему будет доступна для комментирования, если вы не отключите эту возможность для каждой отдельной записи через Консоль. Если вы хотите подойти к решению вопроса еще более радикально, можете удалить файл `wp-comments.php` из ядра WordPress, что физически заблокирует возможность комментирования ваших записей. Впрочем, мы рекомендуем вам использовать более тонкие методы.

Модерация комментариев и CAPTCHA

Один из способов борьбы с нежелательными комментариями — замедление деятельности спамеров. Этот подход достаточно прост, однако он также замедляет и деятельность обычных комментаторов. Вы можете предложить комментаторам регистрироваться на сайте в качестве пользователей для получения доступа к дополнительным возможностям комментирования записей. Этот вариант рассматривается далее в настоящем разделе, однако случайные посетители не смогут делиться своими мыслями на вашем сайте. Также это решение подразумевает ваше активное наблюдение за регистрацией пользователей, так как некоторые пользователи будут регистрироваться на вашем сайте исключительно с целью размещения на нем нежелательной информации.

Другим инструментом для замедления деятельности спамеров без отключения возможности комментирования является модерация. Можно установить модерацию всех комментариев или публиковать без модерации только комментарии от уже проверенных комментаторов. По сути, вы берете на себя обязательства по выявлению спама, отслеживаете каждый новый комментарий и принимаете решение о его публикации или отклонении. И в этом случае кажущийся невинным комментарий может оказаться первым шагом пользователя, стремящегося к размещению нежелательной информации на вашем сайте. По мере усовершенствования механизмов обеспечения безопасности злоумышленники тоже становятся умнее, применяют больше автоматизированных средств и предварительно испытывают защитные системы, которые хотят обойти.

Внесение IP-адресов, с которых поступает нежелательная информация, совмещает в себе жесткие методы и метод модерации. Доступ можно контролировать при помощи файла `.htaccess`. Впрочем, и этот способ чем-то похож на пальбу из гигантского ружья по комарам, ведь вполне возможно, что в итоге вы заблокируете IP-адрес, с которого ваш сайт посещают не только спамеры, но и обычные пользователи. Также помните, что спамеры легко меняют IP-адреса при помощи бот-сетей и других ресурсов, и такой способ борьбы с ними просто не позволит вам окончательно решить эту проблему.

Требование ввода капчи (CAPTCHA)препятствует размещению нежелательных комментариев спамерами, так как от них требуется ввести дополнительную динамически изменяющуюся информацию. Существует немного плагинов, позволяющих использовать ввод капчи при работе с WordPress, и все они предлагают при попытке размещения комментария ввести отображаемое на экране слово или решить математический пример. Наиболее простым плагином является Math Test, который предлагает пользователю решить пример на сложение двух чисел. Основная идея этого метода заключается в том, что процессы автоматического размещения нежелательной информации не могут распознать слова или решить примеры, что предотвращает размещение такой информации на сайте. Нет однозначного мнения по поводу эффективности этого метода, так как уровень его отказа достаточно высок, около 20%, да и комментаторы, желающие вступить в обсуждение, вынуждены выполнять это пусть и простое, но все-таки дополнительное действие. А если ваш

сайт посещают не только англоговорящие посетители, метод ввода капчи, предлагающий английские слова, набранные изогнутым шрифтом, защитит ваш сайт в том числе и от ценных комментариев расстроенных пользователей.

Бесплатный плагин WP-Spam предлагает обратный порядок ввода капчи: этот плагин проверяет, использует пользователь браузер или автоматический процесс. Такой набор приемов JavaScript представляет собой один из способов борьбы с нежелательной информацией, и его эффективность и воздействие на пользователей зависят от территориального размещения посетителей вашего сайта так же, как и в случае со всеми остальными способами.

Автоматизация обнаружения спама

Первым шагом к автоматизации обнаружения спама является занесение определенных типов записей или определенных слов в черный список. Через панель наблюдения выберите пункт Settings (Настройки) ▸ Discussion (Обсуждение) в меню Comment Moderation (Модерация комментариев). Там вы сможете заблокировать размещение любых комментариев, если они содержат больше ссылок, чем вы укажете. Не устанавливайте значение этого параметра равным нулю, так как в этом случае будут заблокированы любые комментарии, в которых пользователи указывают адрес своего сайта. Эта функция позволяет отсеять явные спам-сообщения. Также вы можете добавить в черный список конкретные слова, например «Викодин», что поможет отсеять нежелательные сообщения, рекламирующие поддельные фармацевтические препараты. Впрочем, если вас беспокоят частые предложения поддельных «Роллексов», не вносите в черный список слово «watches» (ведь *watches* это не только часы, но и глагол «смотреть»), поскольку в этом случае все комментарии, в которых это слово используется как глагол, также будут заблокированы. Черные списки конкретных слов эффективны в применении, если в них вносятся слова, которые имеют одно значение независимо от контекста.

К счастью, в систему WordPress встроен плагин Akismet, который позволяет справиться с нежелательными комментариями за счет применения общего черного списка, и при этом принцип его работы очевиден для пользователей. Пройдите по ссылке http://akismet.com/, чтобы зарегистрировать ключ API (интерфейса программирования приложений) для использования этой службы; затем перейдите на консоль и настройте параметры плагина Akismet — это необходимо, чтобы проверить, может ли ваша система WordPress подключиться к службе Akismet. Фактически плагин Akismet сравнивает каждый новый опубликованный комментарий с базой данных по нежелательным комментариям Automattic и принимает решение о том, является ли конкретный комментарий спамом. Согласно статистике, приведенной на сайте akismet.com, до 80% всех комментариев относятся к нежелательным; также утверждается, что при помощи этой службы были выявлены и помечены как спам более 55 триллионов комментариев.

Помимо функций, выполняемых встроенным плагином, служба Akismet используется и другими способами. Akismet работает и с другими системами управления

контентом. Стоимость использования Akismet зависит от объема и типа вашего сайта и может составить от 0 до 50 долларов в месяц. Несмотря на то что мы подчеркиваем бесплатность системы WordPress и большинства связанных с ней тем и плагинов, мы все же рекомендуем воспользоваться платной службой Akismet, обеспечивающей защиту от спама. По сравнению со стоимостью коммерческого хостинга WordPress наиболее бюджетные варианты службы Akismet стоят совсем немного, и всего за несколько долларов в месяц вы можете избежать выполнения достаточно времязатратной административной работы.

Обеспечение безопасности сайта WordPress

К сожалению, становясь более успешными и популярными, вы также становитесь мишенью. WordPress — успешная и популярная платформа для веб-сайтов, что не может не привлекать внимания хакеров и других злоумышленников. Очевидно, что злоумышленники, желающие создать сеть сайтов, будут выискивать наиболее популярные приложения и наносить удары в их уязвимые места. К сожалению, одним из слабых мест системы WordPress (так же, как и PHP) является то, что в связи с низким порогом входа и простотой использования пользователи с небольшим объемом технических знаний и не сильно озабоченные вопросами безопасности используют систему, не до конца понимая всю серьезность вопроса обеспечения безопасности.

В этом разделе рассматриваются основные принципы обеспечения безопасности, рекомендуемые к исполнению при использовании системы WordPress. Некоторые из них продиктованы здравым смыслом, однако, как это ни удивительно, не все среднестатистические сайты применяют их. Все описанные меры являются профилактическими, их следует применять до того, как в них возникнет реальная необходимость. Как говорил Бенджамин Франклин: «Унция профилактики лучше фунта лечения». Время, которое вы потратите на защиту своего сайта, полностью окупится, если вам придется приводить свой подвергшийся вторжению сайт в порядок.

Обновления

Правило № 1 — всегда используйте последние обновления. Разработчики WordPress постоянно работают над улучшением своей системы, стараясь сделать ее более надежной и стабильной. Это одно из ключевых преимуществ программного обеспечения с открытым кодом. Множество разработчиков, каждый из которых обладает разными навыками и умениями, ежедневно пересматривают код системы, проверяют и обновляют его, чтобы улучшить общую кодовую базу.

Обновления часто помогают справиться с непростыми аспектами в области обеспечения безопасности до того, как они станут реальными проблемами на практике. Обычно атаки направлены на необновленные версии WordPress, тогда как регулярно обновляемые сайты в достаточной степени защищены.

WordPress выводит на Консоль уведомления, благодаря которым вы можете узнать о доступности новой версии. Другая новая функция обеспечивает возможность обновления WordPress непосредственно через Консоль. Команда WordPress приложила немало усилий, чтобы сделать процесс обновления максимально безболезненным. Эта новая функция позволяет администратору сайта обновлять ядро WordPress непосредственно через веб-интерфейс.

Если ваш сетевой сервер позволяет записывать файлы в директории WordPress, вы можете использовать функциональные возможности автоматического обновления. В противном случае WordPress запросит данные доступа по FTP для обновления файлов. В обоих случаях возможно возникновение проблем. В общем, ваш веб-пользователь не должен обладать правами записи данных в корневой каталог документов. Это может вызвать больше проблем, чем принести пользы, особенно если речь идет об общей хостинг-платформе. Исключение, разумеется, делается для отдельных директорий, таких как папка «Загрузки», которую веб-пользователь может перезаписывать, так как это обеспечивает функционирование системы.

Кроме того, неясно, каким образом хранятся или используются данные доступа по FTP. Несмотря на низкую вероятность неприятных происшествий, пользователи предпочитают не вводить свои данные FTP (при использовании незащищенного протокола) в какую-либо форму, запрашивающую их. Однако если вы примете решение использовать этот способ, можно прописать определенные конфигурационные переменные WordPress в файле `wp-config`, что поможет в дальнейшем автоматизировать процесс FTP.

Существует определенное равновесие между простотой исключительно важной процедуры обновления ядра WordPress и обеспечением безопасности корневого каталога.

Также существует функция уведомления об обновлениях и предоставления журнала изменений. Уведомления информируют вас о наличии обновлений для ядра WordPress, а также об уже установленных темах и плагинах. Многие обновления сопровождаются журналом изменений, который содержит информацию о том, что именно изменилось в новой версии. Эта функция может оказаться полезной, если вы не можете принять решение о необходимости срочного обновления, — благодаря ей можно принять решение об отсрочке обновления до даты, указанной в графике обслуживания. Предоставление актуальной информации в протоколе изменений зависит от разработчика.

Сокрытие информации о версии WordPress

Этот раздел посвящен сокрытию от общественности информации об используемой версии WordPress. На самом деле существуют разные способы для решения этой задачи. Специалисты по пропагандированию WordPress предлагают не скрывать эту информацию, а, наоборот, выставлять ее на всеобщее обозрение и гордиться ею. С другой стороны, пользователи, обеспокоенные вопросами безопасности, склоняются к тому, что эту информацию все же лучше скрыть. Именно благодаря

этой информации злоумышленники с легкостью находят уязвимые сайты. Зачем выставлять эту информацию напоказ?

С другой стороны, позиция разработчиков WordPress имеет под собой достаточное основание: при поиске уязвимых сайтов через бот-сеть маловероятно, что кто-то будет искать конкретные версии WordPress, ведь бот-сеть может просто атаковать сайт, и для этого потребуется столько же времени, сколько уйдет на поиск информации о версии системы. Зачем же кому-то тратить в два раза больше времени? Если ваш сайт взломают из-за того, что вы используете устаревшую версию WordPress, сокрытие информации о версии системы вас не спасет; в любом случае следует своевременно обновлять свою систему.

При установке стандартного пакета WordPress информация о версии отображается в программном коде HTML в виде метатега, доступного для просмотра каждому, кто просматривает программный код. Впрочем, если вы хотите удалить данный метатег, существует ряд плагинов, которые помогут это сделать. Также вы можете отредактировать файл `functions.php`, добавив в него последнюю строку:

```
remove_action('wp_head', 'wp_generator');
```

И не забудьте: некоторые темы и плагины выводят информацию о версии вашей системы в заголовке страницы на сайте, будьте внимательны.

Ограничение количества попыток входа в систему

Среди прочих мер предосторожности рассмотрим ограничение количества попыток входа в систему через Консоль WordPress. Такая мера предосторожности может предотвратить атаку на ваш сайт методом грубого перебора паролей. По умолчанию система WordPress допускает неограниченное количество попыток входа на сайт, это означает, что автоматический сценарий будет иметь возможность пытаться войти на ваш сайт сутки напролет.

Плагин Limit Login Attempts авторства Ионы Енфелдта может помочь решить эту проблему. После совершения предварительно заданного количества попыток входа на сайт IP-адрес, с которого идут попытки доступа, блокируется на заданный период времени. Такая мера снижает привлекательность вашего сайта для атак при помощи автоматических сценариев. Информация о плагине Limit Login Attempts приведена на сайте http://wordpress.org/extend/plugins/limit-login-attempts/.

Использование надежных паролей

Также рекомендуется использовать надежные пароли для доступа к учетной записи, причем эта рекомендация относится к любым учетным записям в Интернете, а не только к записям в системе WordPress. Да, нам приходится запоминать сотни паролей, однако при помощи мнемосхем и программ для хранения паролей вполне возможно использовать только надежные пароли. В WordPress есть удобный показатель качества паролей JavaScript, который помогает оценить надежность пароля

при его вводе. Хорошим надежным паролем может оказаться и что-то, что вам легко запомнить, а также вы можете использовать приложение для сохранения паролей. Ваш пароль — ключ от вашего королевства, и этот ключ следует выбирать тщательно.

Изменение префикса таблицы

А вот и еще один метод для того, чтобы сделать вектор атаки неявным. По умолчанию префикс таблицы в новых версиях системы WordPress выглядит как `wp_`. Таким образом, любой таблице в базе данных вашей системы WordPress присваивается достаточно предсказуемое имя, что упрощает задачу при выборе типа атаки на ваш сайт для злоумышленников. В случае разработки нового сайта используйте какой-либо уникальный префикс.

Если вы используете уже существующий сайт, вам пригодятся плагины, которые смогут переименовать таблицы в вашей системе. WP-Security Scan авторства Майкла Торберта, который рассматривается далее в этой главе, может изменить префиксы таблиц в вашей системе. До выполнения этой задачи создайте резервную копию базы данных, поскольку в случае неправильного срабатывания плагина последствия могут оказаться достаточно неприятными.

Перемещение файла конфигурации

По умолчанию файл конфигурации WordPress размещается в корневом каталоге вашего веб-сайта. В случае прекращения работы PHP на вашем сервере, независимо от причин этого происшествия, возможно отображение этого файла в нешифрованном текстовом формате, что делает доступными ваши пароли и информацию о вашей базе данных.

Существует безопасный способ перемещения директории `wp-config` из корневого каталога, что позволит предотвратить ее случайное отображение. WordPress имеет встроенную функцию для автоматической проверки директории предыдущего уровня в случае, если файл конфигурации не найден.

Однако в некоторых ситуациях при использовании определенных хост-систем такой вариант решения проблемы невозможен. В качестве альтернативы предлагается настроить `.htaccess` таким образом, чтобы он не обслуживал файл `wp-config`. Для этого в файл `.htaccess`, расположенный в корневом каталоге, следует добавить строку кода:

```
<FilesMatch ^wp-config.php$>deny from all</FilesMatch>
```

Перемещение директории с контентом

Начиная с версии WordPress 2.6 вы можете перемещать директорию `wp-content`. Это позволит перенести существенную часть системы WordPress в непредусмотренное по умолчанию место. Также это создаст определенные сложности для злоумышленников и, возможно, убедит их оставить ваш сайт в покое.

Добавьте две строки кода в файл `wp-config`:

```
define('WP_CONTENT_DIR',
   $_SERVER['DOCUMENT_ROOT'].'/mysite/wp-content');
define('WP_CONTENT_URL',
   'http://example.com/mysite/wp-content');
```

При использовании ряда плагинов и нестандартной структуры каталогов могут возникнуть определенные сложности. В случае возникновения подобных проблем добавьте приведенные ниже строки кода в файл `wp-config`, что обеспечит необходимый уровень совместимости:

```
define('WP_PLUGIN_DIR',
   $_SERVER['DOCUMENT_ROOT']. '/mysite/wp-content/plugins');
define( 'WP_PLUGIN_URL',
   'http://example.com/mysite/wp-content/plugins');
```

Само по себе перемещение директории с контентом не повышает уровень безопасности вашего сайта, а всего лишь предотвращает возможность атаки на ваш сайт при помощи автоматизированных средств. Подобные автоматизированные средства ищут сайты с наименьшим общим знаменателем, и потому они ищут стандартные конфигурации WordPress с настройками, предусмотренными по умолчанию, поскольку их проще всего взломать. Обеспечение безопасности за счет скрытия информации, по сути, не является обеспечением безопасности, однако снижает привлекательность вашего сайта для злоумышленников.

Использование функции «Секретный ключ»

Файл `config` в вашей системе WordPress содержит значения секретных ключей для шифрования куки пользователей. Начиная с версии WordPress 2.6 используются четыре ключа для выбора секретного, или частного, ключа, который WordPress использует для защиты информации о сеансе, которая хранится в куки пользователя. У каждого ключа есть модификатор «соль», который используется при шифровании для снижения вероятности того, что атака на основе каталогов приведет к подбору пароля методом перебора. Для начала атаки потребуется подобрать и пароль, и модификатор «соль». Если вы не указываете модификаторы «соль», их самостоятельно создает WordPress.

Для более качественного шифрования данных о сеансе пользователя следует задать и секретный ключ, и модификатор «соль». Либо придумайте их, либо посетите сайт https://api.wordpress.org/secret-key/1.1/salt и используйте случайно генерируемые ключи.

Ключи можно изменить в любое время, однако при их изменении пользователи, уже авторизованные на сайте, будут вынуждены повторно совершить вход на сайт.

```
define('AUTH_KEY',
   'C?m92_K%B[7,,Onl(&WG,oodC9ue1y;aUK[e,.E+([Y?0D+/ ]i*!PkF?I:U+C^6');
define('SECURE_AUTH_KEY',
   'oesV)E.Z<y@o.o1eM|c@7)t^SL:06WjDENo;t_j.e4(eX@8#`~gy1S&R*Gf!k19+');
```

```
define('LOGGED_IN_KEY',
   'Y?fgU+EleuDKE3n-^~cF%IbgTR,ep+UZE={>8,j,EO+7a-u|c]EH;|G@|4ZS#a+-');
define('NONCE_KEY',
   'g6f<q6QB| 1u59Q]~(r1B@<dl2f]rkQVg7HMx}!B#OzPPyG[.N{RV<yA2l+=.r7#');
define('AUTH_SALT',
   '}e@Vd[W0d}?u&Ps>MyO1NZT>tU[Kg4QW%+y-fyRU|d-PWAV7az+a0K6qx-{iwSv)');
define('SECURE_AUTH_SALT',
   '9Nbtg_v1D}?f/rj*/p;a[)}jq-y&YVxqA6.KSk;am:sjH}-!uN6n5]i?NIuW&9<l');
define('LOGGED_IN_SALT',
   'pQ;TXDxRN`TmDl$+gU0EgG-1OMYM*[p6R}07)7Fs*/%Yec]t|E+piqLf1.t2kLTc');
define('NONCE_SALT',
   '~<A7*5IS&N:Gy!:yYM`LuggB0^1RIjSy:QEOP@.TZs!Dq-73i3KQYa:3j1WYIeUg');
```

Не используйте приведенные значения, укажите свои собственные.

Принудительное использование SSL при входе в систему и администрировании

Вы можете вынудить своих посетителей и администраторов входить на сайт через зашифрованную страницу SSL при условии, что вы ее предварительно настроили. Отредактируйте файл `config` в системе WordPress, добавив признак:

```
define('FORCE_SSL_LOGIN', true);
```

Также вы можете установить доступ к Консоли WordPress при помощи HTTPS. Для этого, опять же, потребуется отредактировать файл `config`, добавив в него строку:

```
define('FORCE_SSL_ADMIN', true);
```

Пожалуйста, не включайте эти функции «наобум». Если на сайте используется самозаверенный сертификат, это может повлечь за собой определенные сложности. Обратите внимание на то, что в системе WordPress принято устанавливать внутренние ссылки при помощи URL, через которые вы переходите на консоль. Таким образом, если вы принудительно запустили SSL на консоли и используете самозаверенный сертификат сервера, внутренние ссылки URL также примут этот сертификат, и ваш посетитель будет вынужден принять его. Это считается неправильным. Обсудите со своим хостинг-провайдером возможность получения сертификата от признанного поставщика сертификатов.

Разрешения Apache

Разрешения зависят от конфигурации вашей системы, однако рекомендуется установить разрешения для файлов 644, а для директорий — 755. Если вы не можете загружать файлы в папку «Загрузки», устанавливайте только такие привилегии. Обычно файлы относятся к той же группе, что и веб-сервер, а их владельцем выступает локальный пользователь, например:

```
drwxr-xr-x  8 ddamstra www-data      4096 2012-09-02 16:34 .
drwxr-xr-x  8 ddamstra root          4096 2010-12-18 20:39 ..
```

```
-rw-r--r--  1 ddamstra www-data     9835 2011-03-15 13:17 apple-touch-icon.png
-rw-r--r--  1 ddamstra www-data    41662 2010-12-07 15:04 favicon.ico
-rwxrwxr-x  1 ddamstra www-data     3456 2010-12-07 14:51 .htaccess
-rw-r--r--  1 ddamstra www-data      395 2012-09-02 16:34 index.php
-rw-r--r--  1 ddamstra www-data     9177 2012-09-02 16:34 readme.html
-rwxrwxr-x  1 ddamstra www-data     1137 2011-07-13 09:07 sitemap.xml
-rwxrwxr-x  1 ddamstra www-data      514 2011-07-13 09:07 sitemap.xml.gz
drwxr-xr-x  6 ddamstra www-data     4096 2012-09-02 16:35 .svn
-rw-r--r--  1 ddamstra www-data     4264 2012-09-02 16:34 wp-activate.php
drwxr-xr-x 10 ddamstra www-data     4096 2012-09-02 16:34 wp-admin
-rw-r--r--  1 ddamstra www-data     1354 2012-09-02 16:34 wp-app.php
-rw-r--r--  1 ddamstra www-data      271 2012-09-02 16:34 wp-blog-header.php
-rw-r--r--  1 ddamstra www-data     3522 2012-09-02 16:34 wp-comments-post.php
-rw-r--r--  1 ddamstra www-data     3177 2011-02-24 09:15 wp-config-sample.php
drwxr-xr-x  7 ddamstra www-data     4096 2010-11-30 15:57 wp-content
-rw-r--r--  1 ddamstra www-data     2726 2012-09-02 16:34 wp-cron.php
drwxr-xr-x  9 ddamstra www-data     4096 2012-09-02 16:34 wp-includes
-rw-r--r--  1 ddamstra www-data     1997 2011-02-24 09:15 wp-links-opml.php
-rw-r--r--  1 ddamstra www-data     2341 2012-09-02 16:34 wp-load.php
-rw-r--r--  1 ddamstra www-data    29084 2012-09-02 16:34 wp-login.php
-rw-r--r--  1 ddamstra www-data     7712 2012-09-02 16:34 wp-mail.php
-rw-r--r--  1 ddamstra www-data     9916 2012-09-02 16:34 wp-settings.php
-rw-r--r--  1 ddamstra www-data    18299 2012-09-02 16:34 wp-signup.php
-rw-r--r--  1 ddamstra www-data     3700 2012-09-02 16:34 wp-trackback.php
-rw-r--r--  1 ddamstra www-data     2788 2012-09-02 16:34 xmlrpc.php
```

Обратите внимание, что в этом случае возможно нарушение ряда удобных функций системы, например обновления одним щелчком мыши, равно как и функций тем и плагинов, установленных через панель управления. Для восстановления этих функций может потребоваться предоставить WordPress данные доступа FTP по сайту. В разделе «Обновления» приведена более подробная информация по этому вопросу.

Имя пользователя и пароль MySQL

Правильно выбирайте свой логин MySQL и назначайте уровни доступа. Ни в коем случае не подключайте свой сайт WordPress к базе данных через привилегированного пользователя MySQL. Для каждого из сайтов WordPress следует создавать отдельного пользователя. Убедитесь в том, что ему предоставлены только необходимые права доступа к базе данных. Например, пользователь базы данных вашего сайта WordPress не должен иметь доступ к данным другого пользователя.

Рекомендованные плагины для обеспечения безопасности

Для обеспечения безопасности нужно быть начеку. Сложно ожидать, что что-то, к чему вы не относитесь с достаточным вниманием, будет работать. Ряд плагинов поможет вам с обслуживанием систем безопасности и настройкой вашей системы WordPress. Аналогично принципам работы антивирусных систем и систем

обнаружения вредоносного программного обеспечения на рабочих станциях описанные ниже инструменты могут помочь вам защитить систему.

WP-Security Scan

WP-Security Scan, разработанный Майклом Торбертом, как показано на рис. 13.5, обеспечивает общую проверку безопасности вашей системы WordPress. Этот плагин позволяет проверить многие из описанных ранее параметров, в том числе версию WordPress, префикс таблиц и отсутствие учетной записи администратора. Также в состав плагина входят средства для анализа файловой системы, которые проверяют, установлены ли разрешения рекомендованным образом. Плагин WP-Security Scan — удобное средство для проверки соответствия базовых настроек принципам обеспечения безопасности. Мы надеемся, что в последующих версиях Торберт добавит новые функции.

Более подробная информация о WP-Security Scan приведена по адресу http://wordpress.org/extend/plugins/wp-security-scan/.

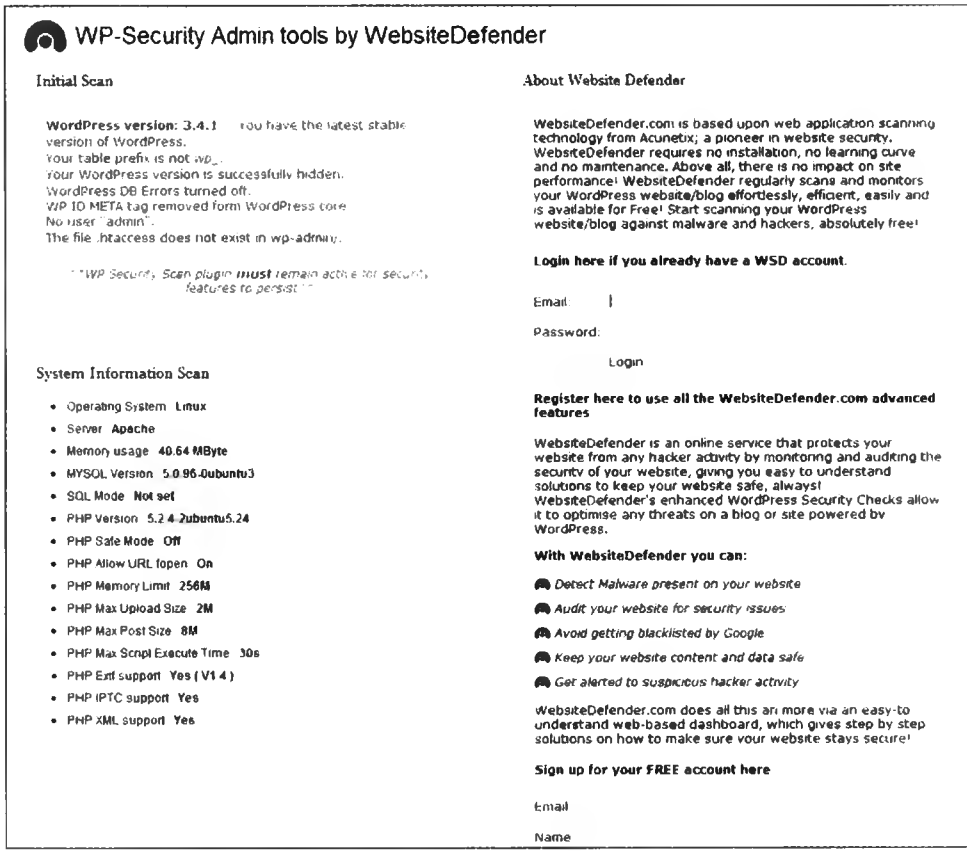


Рис. 13.5. Результат проверки при помощи WP-Security Scan

WordPress Exploit Scanner

WP-Exploit Scanner — еще один плагин Доннча Окайома. Этот плагин позволяет анализировать файлы, записи и комментарии на предмет содержания в них подозрительных данных. По сути, этот инструмент позволяет проанализировать подверженность вашего сайта риску. Этот плагин не удаляет данные, а составляет список подозрительного содержимого, который вы можете изучить. Основная сложность при работе с этим плагином заключается в том, что, даже имея на руках список отфильтрованных данных, вы должны понимать, что именно вы ищете. При использовании этого плагина возможны ложноположительные результаты, которые возникают в связи с плагинами на JavaScript и файлами ядра WordPress.

Вы можете узнать больше о WordPress Exploit Scanner, пройдя по ссылке http://wordpress.org/extend/plugins/exploit-scanner/.

WordPress File Monitor

Плагин WordPress File Monitor авторства Мэтта Уолтерса, представленный на рис. 13.6, выполняет поиск добавленных, измененных или удаленных файлов в вашей

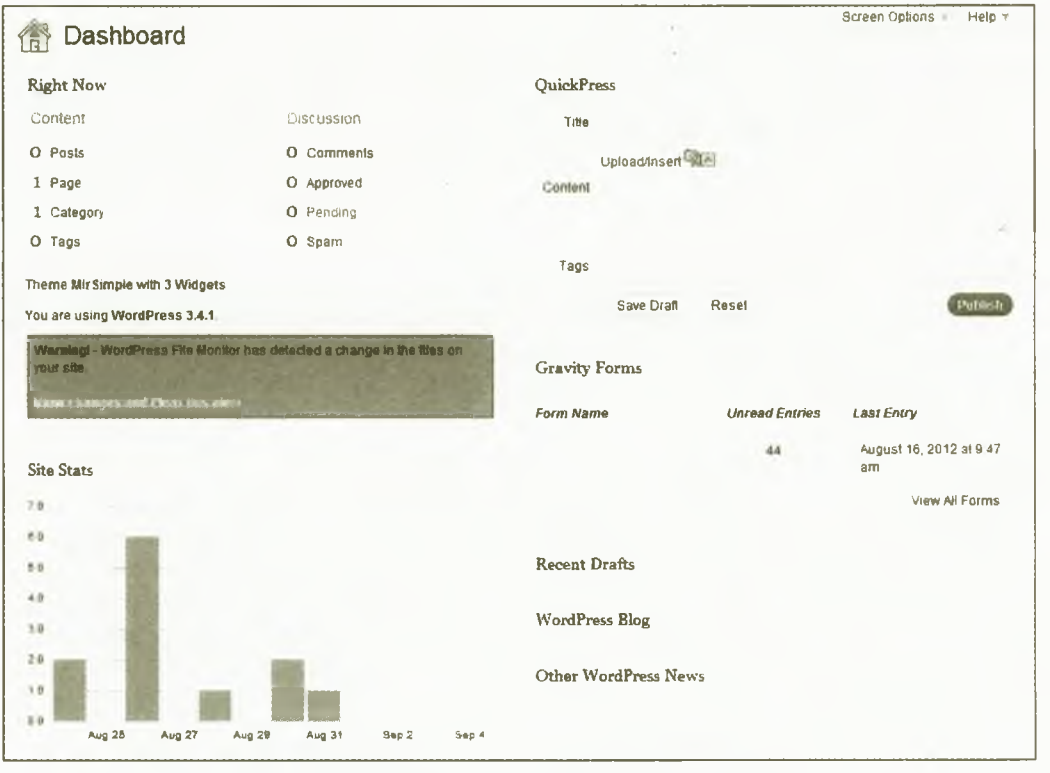


Рис. 13.6. Сообщение от WordPress File Monitor

системе WordPress. Можно настроить плагин таким образом, чтобы каждый раз при обнаружении активности в файловой системе он отправлял уведомление по электронной почте. Также можно исключить определенные директории, такие как папка «Загрузки» или папки кэша на базе файлов, из поиска.

В случае внесения изменений в файловую систему этот плагин направляет предупреждение на Консоль WordPress, а также отправляет вам письмо о происшедших изменениях по электронной почте. Это очень удобно, так как позволяет своевременно отследить нежелательную активность, однако при обновлении системы плагин передает ложные сигналы тревоги. Предполагается, что вы знаете, когда происходит обновление системы, и потому сможете отличить ложные сигналы тревоги от тех, на которые следует обратить внимание.

Более подробная информация о WordPress File Monitor представлена по адресу http://wordpress.org/extend/plugins/wordpress-file-monitor/. Этот плагин не обновлялся в течение некоторого времени, однако мы по-прежнему используем его для наших мультисайтовых проектов.

WordFence Security

WordFence Security авторства Марка Маундера представляет собой полноценный плагин для обеспечения безопасности. Этот плагин проверяет установленные файлы ядра и сравнивает их с текущей версией, хранящейся в информационном архиве WordPress. Этот плагин проверяет файлы ядра. Таким образом, проверяются файлы WordPress и файлы плагина. Тогда как плагин WordPress File Monitor выявляет новые файлы, установленные в корневой каталог вашего сайта, которые потенциально могут быть установлены злоумышленниками, плагин WordFence Security оценивает файлы ядра и плагина на предмет изменения или отклонения кода в них от исходного кода.

Также WordFence анализирует фактическое содержимое на предмет содержания в нем вредоносного кода и подписей, нацеленных на «выуживание» паролей, как показано на рис. 13.7. Кроме того, плагин обладает и другими функциями в области обеспечения безопасности, что позволяет охватить широкий диапазон потенциальных угроз и векторов атаки.

Одной из наиболее интересных функций WordFence является предоставление информации о реальном трафике, который относится к брандмауэру на уровне программного обеспечения. Используя информацию о реальном трафике и функции брандмауэра, вы сможете блокировать доступ на сайт с отдельных IP-адресов или из конкретных стран. Эта функция может пригодиться в случае попытки злоумышленника проникнуть на ваш сайт. Кроме того, вы можете настроить WordFence определенным образом, который позволит автоматически блокировать или прервать попытку доступа с IP-адреса, если источник трафика выходит за установленные вами пределы.

Более подробная информация о WordFence Security приведена на сайте http://wordpress.org/extend/plugins/fence/.

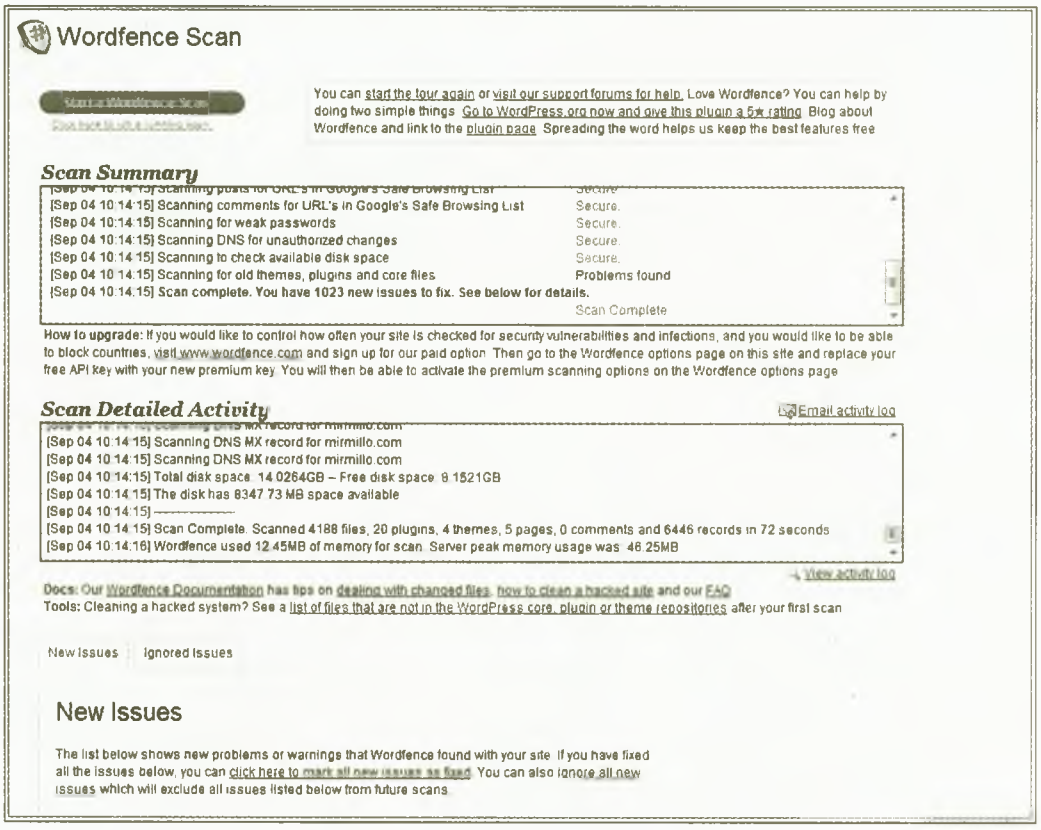


Рис. 13.7. Результаты работы плагина WordFence Scan

Использование ролей в WordPress

Любой обладающий здравым смыслом руководитель умеет распределять задачи между сотрудниками, и администратору сайта также придется освоить это полезное умение. Система ролей в WordPress позволяет предоставлять разные уровни доступа учетным записям пользователей. Предусмотренные по умолчанию роли WordPress охватывают основные уровни доступа и позволяют начать процесс публикации на сайте благодаря базовым возможностям каждой из ролей. Для создания новых ролей с дополнительными возможностями для решения конкретных задач можно использовать различные плагины.

Если вы — единственный администратор сайта, то, вероятно, вам не так уж необходима система ролей. Все роли распределяются между вами — администратором сайта — и ими — массой незарегистрированных пользователей. Такой способ администрирования при отключенной возможности регистрации новых пользователей позволяет вам использовать свой сайт безопасно, однако при этом у пользователей зачастую пропадает желание принимать участие в вашем проекте. Со временем вы можете принять решение о предоставлении постоянным посетителям возможности

входа на сайт, что даст им ряд дополнительных преимуществ с точки зрения простоты использования вашего сайта.

Роли присваиваются пользователям через панель управления. Каждому пользователю должна быть присвоена определенная роль, при этом следует предусмотреть роль по умолчанию, которая будет присваиваться любому зарегистрированному на сайте пользователю. Конкретные роли назначаются через панель управления в зависимости от ваших текущих потребностей и разрешений системы безопасности, которые требуется предоставить.

Роль: Подписчик

Роль подписчика, в целом, предоставляет такие же права доступа, которыми обладает пользователь, не совершивший вход на сайт. Для чего же нужна эта роль, если пользователь, которому она присвоена, может читать записи и комментировать их так же, как и незарегистрированный посетитель сайта?

Эта роль может пригодиться по нескольким причинам. Во-первых, она может использоваться для пользователей, регулярно посещающих ваш сайт. Если эти пользователи зарегистрированы на сайте, от них не требуется каждый раз заполнять все поля формы при отправке комментария. Во-вторых, эта роль позволяет контролировать нежелательные сообщения, так как вы можете разрешить комментирование только для зарегистрированных посетителей и тем самым отсеять множество автоматических роботов, рассылающих спам. Наконец, для использования функциональных возможностей ряда плагинов требуется такой минимальный уровень доступа.

Роль: Участник

Следующей ступенькой является роль участника, которая позволяет вам передать часть своих обязанностей пользователям своего сайта. Основной привилегией участников является возможность создания новых записей, однако они не имеют права публиковать такие записи на сайте, для публикации требуется роль более высокого уровня. Эта роль позволяет пользователям вносить свой информационный вклад в содержимое вашего сайта, однако вы по-прежнему полностью контролируете размещаемую на сайте информацию. Таким образом, эта роль подходит для присвоения пользователям, которые только начинают осваивать публикацию контента.

Помимо создания черновиков записей участники могут также в любое время редактировать свои записи и удалять свои неопубликованные записи. Участники не могут загружать файлы и изображения, которые будут использоваться в их записях.

Роль: Автор

Следующей ступенью в ролевой иерархии является роль автора. Авторам оказывается большее доверие, чем участникам, поэтому они могут загружать файлы, которые будут использованы в их записях, и могут публиковать свои записи без

согласования с администратором сайта. Также авторы могут редактировать и удалять свои собственные опубликованные записи.

Авторы могут работать только со своими записями, то есть они могут читать и комментировать любые другие записи, как и все остальные пользователи, однако изменять они могут только свой собственный контент.

Роль: Редактор

Для роли редактора предусматриваются две новые возможности. До этой ступени иерархии пользователи были ограничены работой с записями, но редактор может работать непосредственно со страницами. Кроме того, редактор имеет право изменять любой контент на сайте.

Эта роль не позволяет управлять учетными записями пользователей или настройками сайта, например темами и плагинами, однако предоставляет полный доступ к действительному контенту. На практике эта роль присваивается пользователю конкретной установки WordPress. Она дает достаточно возможностей для управления ежедневным контентом, представленным на сайте, однако не позволяет пользователю баловаться с общими настройками сайта и нарушать его работу.

Обратите внимание на то, что редакторы не имеют доступа к управлению меню и областям графических элементов, что ограничивает объем контента, который они могут редактировать.

Роль: Администратор

Роль администратора относится к ролям корневого уровня. Администратор имеет доступ ко всей информации, выводимой на Консоль, поэтому будьте осторожны, присваивая эту роль кому-либо из пользователей. Администратор может изменять учетные записи пользователей, темы, плагины и любой контент.

Убедитесь в том, что роли администраторов присваиваются серьезным и ответственным пользователям, которые используют надежные пароли. В случае, если злоумышленники получат доступ к учетной записи администратора вашего сайта, они автоматически получат доступ и к вашему сайту.

Роль: Суперадминистратор

Эта роль используется только в мультисайтовых сетях. Основной задачей этой роли является управление различными сайтами WordPress, входящими в состав сети, а также управление сетью в целом.

Обзор ролей

В табл. 13.1 в упрощенной форме представлен обзор возможностей, предусмотренных для каждой из ролей. Более подробная информация о возможностях каждой

роли приводится в Кодексе WordPress, расположенном по адресу http://codex.wordpress.org/Roles_and_Capabilities.

Таблица 13.1. Возможности ролей, предлагаемых WordPress

Возможность	Суперадминистратор	Администратор	Редактор	Автор	Участник	Подписчик
Управление сетью	X					
Управление темами	X	X				
Управление плагинами	X	X				
Управление пользователями	X	X				
Управление возможностями сайта	X	X				
Модерирование комментариев	X	X	X			
Управление категориями	X	X	X			
Управление ссылками	X	X	X			
Управление всеми записями	X	X	X			
Управление всеми страницами	X	X	X			
Управление чужими записями	X	X	X			
Чтение и управление личными записями	X	X	X			
Чтение и управление личными страницами	X	X	X			
Загрузка файлов	X	X	X	X		
Публикация записей	X	X	X	X		
Удаление собственных опубликованных записей	X	X	X	X		
Редактирование собственных записей	X	X	X	X	X	
Удаление собственных неопубликванных записей	X	X	X	X	X	
Чтение	X	X	X	X	X	X

Дополнительные роли

В большинстве случаев ролей, предусмотренных по умолчанию, достаточно, однако в определенных обстоятельствах может потребоваться расширить возможности для определенных ролей, предоставив конкретным пользователям больше разрешений или возможностей для управления контентом.

Плагин Role Scoper авторства Кевина Беренса (http://wordpress.org/extend/plugins/role-scoper/) представляет собой исключительно мощный инструмент для управления подобными случаями, связанными с разграничением доступа. При помощи этого плагина можно изменять права, предоставляемые пользователям, относительно рассмотренных ранее предоставляемых по умолчанию прав, с избирательным контролем доступа в зависимости от установок, определяемых контентом.

Таким образом, пользователь любого уровня может получить дополнительные права для редактирования и управления контентом — страницами или записями — на основании определенных категорий. Этот плагин позволяет как предоставить пользователю дополнительные права на изменение контента, так и ограничить пользователя в правах, например запретить для конкретной роли возможность ознакомления с содержимым.

Например, вы создали сайт, на котором представлены различные линейки программных продуктов. За контент каждой линейки отвечает администратор. В этом случае каждая линейка программных продуктов рассматривается как рубрика WordPress. При помощи этого плагина вы можете ограничить права администратора линейки конкретных продуктов до размещения нового контента в контролируемой им категории.

Плагин Role Scoper — очень мощный инструмент, который позволяет точно разграничивать полномочия пользователей, однако для решения конкретной задачи могут использоваться и другие, менее функциональные плагины, позволяющие другими способами изменять роли, предусмотренные в WordPress.

Резюме

В этой главе были рассмотрены некоторые проблемы, с которыми вы можете столкнуться по мере возрастания популярности вашего сайта WordPress в Интернете, в том числе такие вопросы, как управление масштабируемостью и эффективностью сайта за счет кэширования и выравнивания нагрузки. По мере возрастания популярности сайта он начинает привлекать внимание спамеров и других злоумышленников, и потому в этой главе также рассматривались некоторые способы борьбы с нежелательной информацией и методы обеспечения безопасности вашего сайта WordPress.

Наконец, была рассмотрена система ролей, которая имеет ключевое значение для контроля рабочего процесса и является основой, обеспечивающей использование WordPress в качестве полноценной системы управления контентом, в том числе при использовании этой системы в составе корпоративных программных приложений. Эти темы будут рассматриваться и в следующей главе.

WordPress как система управления контентом

14

В этой главе:

- Анализ задач системы управления контентом, легко решаемых с помощью WordPress
- Настройка WordPress для работы с организацией и отображением более сложного контента
- Интеграция интерактивных элементов, таких как формы, сообщения электронной почты и корзины

Вопрос использования WordPress как системы управления контентом (*англ.* content management system, CMS), похоже, ежемесячно поднимается в Сети. Введите этот вопрос в поисковик, и вы немедленно получите бесчисленные советы о том, «почему», «почему нет» и «как». Кажется, что за WordPress закрепилось звание «всего лишь» движка для блогов, хотя, как вы уже знаете, он предоставляет намного более широкие возможности. С момента первого издания этой книги данная тема и связанные с ней дискуссии также существенно расширились. Использование WordPress больше не ограничивают сегментом «движка для блогов», как это было раньше.

Настоящая глава определяет управление контентом с точки зрения перспективы использования WordPress, рассматривает ключевые функциональные области, связанные с CMS, показывает, как реализовать их с помощью WordPress, и, наконец, указывает на области, в которых, несмотря на свою гибкость и функциональность, WordPress не сможет служить оптимальным решением.

Управление контентом

Точно определить, что означает «управление контетом», очень сложно из-за применения этого понятия к довольно широкому кругу программных инструментов

и систем. На одном полюсе находятся Wiki с их полноценным управлением обновлениями и возможностью редактирования от лица нескольких авторов, но с практически отсутствующими механизмами вывода информации, минимальными организацией страниц и навигацией. На другом полюсе — коммерческие программные пакеты, ориентированные на предприятие, позволяющие контролировать доступ, оценивать эффективность, обладающие функциями хранилища и общего доступа к корпоративной документации. Существует четкая разница между областью «управления бизнес-контентом» корпоративного документооборота и самостоятельной издательской деятельностью, однако попытка «привязать» понятие управления контентом только к одной из этих областей приводит к игнорированию богатства выбора программных инструментов в заявленных сферах. Таким образом, с ростом количества простых в использовании и дешевых сетевых инструментов понятие управления контентом все больше соотносится с системами, используемыми для создания коммерческих интернет-сайтов, содержащих онлайн-каталоги и позволяющих взаимодействовать с заказчиком.

Куда же в этом широком диапазоне попадает WordPress? В самом узком смысле движки для блогов являются формой CMS, работающей с минимальным количеством видов содержимого (страницы и обсуждения) в хронологическом порядке вывода. Несмотря на то что WordPress начиналась как система для ведения блогов и именно через эту узкую призму зачастую рассматривает ее общественное мнение, она обладает возможностями, гибкостью и ресурсами для выполнения большинства, если не всех, требований, обычно предъявляемых рынком к системам управления контентом. Механизмы управления сайтом, администрирования прав пользователей и группировки данных для их структуризации и распределения не являются специфическими для блогов или других видов контента; они требуют настройки, планирования и создания многоцелевой системы делегирования полномочий. Мы надеемся, что в данной книге нам удалось определить функции WordPress по «управлению контентом», поэтому теперь мы можем сложить все элементы мозаики и переключиться на более общее рассмотрение CMS.

Ниже приведены характеристики CMS, рассматриваемые в данной главе:

- **Схема работы и делегирование.** Зачастую ключевой проблемой в мире CMS является реализация контроля за процессом публикации и редактирования для нескольких авторов, обладающих минимальными техническими знаниями. WordPress облегчает добавление контента и управление его распределением для пользователей без технических знаний.
- **Организация контента.** Начиная с эмуляции простейших сетевых порталов и заканчивая созданием сложных иерархий страниц, организация контента всегда связана с управлением множеством различных типов содержимого и выбором подходящей схемы отображения для каждого из них.
- **Интерактивность.** Списки электронных рассылок, формы, дискуссии и функционал для электронной коммерции являются обычными функциями, требующими некоторого расширения WordPress.

- **Другие системы управления контентом.** Будучи полноценной системой управления сайтами, WordPress может служить мощной платформой для создания и редактирования контента, а также переноса его в другие аналогичные системы, такие как Drupal. В данной главе также рассматриваются области, для которых WordPress не является оптимальным выбором.

По своей сути ведение блогов и управление контентом может быть связано с самыми разными источниками, в которых уже существует сложившийся лексикон, однако их расширяемость, возможность персонализации дизайна, а также общение в сообществе разработчиков WordPress привели к размыванию границ между «простым ведением блогов» и новомодным «управлением корпоративным контентом». Можно привести список причин, по которым стоит воспринимать WordPress как превосходную систему управления контентом:

- **Простота.** На всех уровнях, от интерфейса пользователя до создания контента, WordPress может быть простым или сложным в зависимости от навыков пользователя и требований внедрения.
- **Гибкость.** WordPress работает с множеством архетипов, от простых записей в блогах задним числом до многопользовательских гиперлокальных новостных сайтов (см. http://injersey.com) и вывода иерархически организованного статичного и динамического содержимого для художников, фотографов и других творческих людей, а также для рекламы мест, где можно провести отпуск (см. http://baja.com).
- **Расширяемость.** Вы можете найти плагины и темы для создания огромного числа визуальных стилей, интегрировать бесконечное количество типов содержимого и источников, а также упростить процесс перехода от простого контента к сложному, отображаемому в HTML.

Целью рассмотрения WordPress в качестве CMS является анализ типичных проблем управления контентом, техник построения материала и примеров, приведенных в предыдущих главах. Само собой, нежелательно, чтобы каждый диалог начинался с выступления в защиту WordPress как со стороны автора, так и с вашей стороны в ситуации, когда вы выбираете инструменты управления содержимым. Каким бы ни было ваше определение управления контентом или же ваши цели при создании сайта, существенно превосходящего банальный перечень записей в блогах, процесс управления содержимым начинается с упрощения схемы работ.

Рабочие процессы и делегирование

Одним из основных достоинств классической CMS является то, что она упрощает создание и управление содержимым. Это четко связано с разделением полномочий, так как и пользователи, и администраторы с правом редактирования содержимого имеют доступ и несут ответственность за все, что публикуется с помощью CMS.

Пользовательские роли и делегирование

Управление пользователями в CMS связано с настолько же сложным разделением полномочий и разработкой политик, что и деятельность правительства или органов государственной власти. Вам необходимо распределить роли на основе предполагаемых типов категорий контента, а также задать ограничения возможностей пользователей по публикации или редактированию ранее опубликованного содержимого. В чистой среде многопользовательского сайта различия могут показаться не столь важными, но при использовании WordPress в качестве визитной карточки для сетевой торговли или для создания каталога продукции компании обычно требуется участие множества подразделений и контролирующих инстанций.

Плагин Кевина Беренса Role Scoper рассматривался в главе 13 в рамках дискуссии о безопасности и управлении пользователями. Этот плагин позволяет вам создавать новые роли для ваших пользователей и наделять их новыми, весьма детализованными полномочиями. В среде CMS это позволит делегировать создание контента разным подразделениям и разрешать им внесение изменений только в конкретные области.

Наделение полномочиями восходит по иерархии пользователей, а не нисходит от редактора к отдельному автору. В стандартных издательских средах редактор имеет возможность выделять работу для писателей и литературных экспертов, создавая схему работы над конечным продуктом в виде дерева, напоминающего структуру организации. WordPress позволяет задействовать «листья» этого дерева: каждый пользователь, обладающий правами участника или автора, может создавать контент (и в случае с авторами — загружать файлы) и управлять публикацией своих записей. Решение вопроса о том, как и где разделить ответственность, является базовым для создания основы CMS с помощью WordPress:

- Будьте аккуратны с правами администраторов. Раздайте для них пароли уровня `root` или `sudo`. В то же время не путайте редакторов с администраторами. Редакторы могут изменять способ вывода страницы или как-то иначе связывать содержимое. Администраторы же редактируют темы, файлы ядра и плагины. Каждый из них должен быть осведомлен о границах своей зоны доступа.
- Обращайтесь с редакторами правильно. Они должны иметь доступ к редактированию страниц, изменению содержимого или статуса любой записи, а также к изменению метаданных на сайте WordPress. Они также должны использовать свои роли для управления работой авторов и участников.
- Если вы хотите, чтобы каждый элемент содержимого исправлялся до поступления в общую сеть, необходимо отделить участников (не имеющих права на публикацию) от авторов (имеющих право публиковать свои работы, но не редактировать чужие). Создание класса пользователей «участники», с одной стороны, позволяет обеспечить ваших редакторов работой, но с другой — полностью передает им право принятия решения о публикации.

- Обратите внимание на то, что роли и делегирование касаются творческого процесса, а не управления доступом к уже опубликованному контенту. С момента, когда запись становится доступной, и до момента ее удаления (хотя даже после этого она может быть закэширована или воспроизведена из потока где угодно) она принадлежит общественности. В противоположность другим системам управления контентом WordPress не фокусируется на механизмах контроля доступа к опубликованному контенту; не касается управления интеллектуальной собственностью, ведь даже хранилище корпоративной документации позволяет отслеживать доступ, ссылки и содержит механизмы их проверки.

На стыке между многоцелевой и многопользовательской платформами WordPress находится WordPress Multisite, рассматриваемый в главе 10. На WordPress Multisite построен сайт WordPress.com, так как именно эта версия позволяет нескольким независимым группам пользователей запускать независимые, но совместно размещаемые сайты. Это может показаться привлекательным в том случае, если у вас имеются независимые группы товаров или несколько брендов, каждый из которых нуждается в отдельной сборке WordPress, а вы ограничены (или хотите установить ограничения) в плане полномочий администрирующего персонала.

Рабочий процесс

После создания пользователей и их ролей следующим шагом является создание четкой рабочей схемы для переноса контента из человеческих мозгов в Сеть. По окончании краткого анализа структуры пользователей «рабочий процесс», вероятно, должен стать следующим термином, который пользователи часто ассоциируют с популярными CMS. В рамках WordPress существуют два основных компонента рабочего процесса: управление записями и история редакций записей.

История исправлений отображается в секторе записей в Консоли, где при входе в режим редактирования конкретной записи выводится список ее редакций. Если вы запускаете систему с несколькими авторами и редакторами, где редакторы имеют возможность совершенствовать оригинальное творчество авторов, убедитесь в том, что редактирующий персонал использует функцию редакций, что позволяет отследить добавление или сокращение содержимого, а также отличия между различными исправлениями одной записи. Это послужит эффективным инструментом управления исходным кодом для содержимого записей, контролируемых в рамках баз данных MySQL для WordPress.

Тем не менее многие люди являются поклонниками простых панелей инструментов, считая их компактными и мощными, и многие администраторы сайтов могут посчитать интерфейс слишком сложным с учетом технических знаний выбранного ими редактора или администратора. Некоторые столбенеют при столкновении со слишком широким кругом возможностей или выбора. Вы можете воспользоваться плагином WP-CMS Post Control Джонатана Аллбута (http://wp-cms.com/our-wordpress-plugins/wp-cms-post-control-plugin/) для отключения ненужных настроек.

Данный плагин устанавливает новую панель инструментов, которая позволяет настраивать панель Write (Написать) на отображение только выбранных полей, как показано на рис. 14.1.

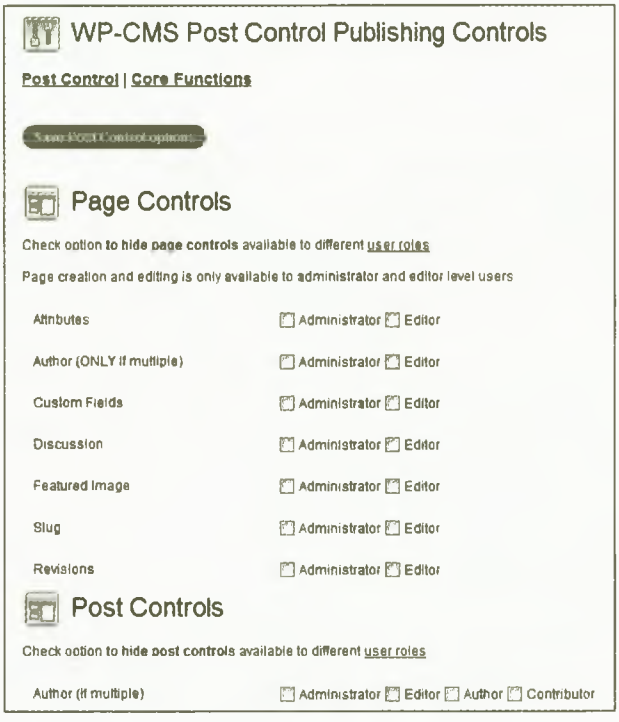


Рис. 14.1. Использование WP-CMS Post Control для настройки возможностей Консоли

Плагин предоставляет множество возможностей, и в то же время является простым инструментом контроля управления. Использование этого плагина — еще один шаг в контроле управления, направленный на создание более простых (или детализованных) панелей редактирования записей, которые будут описаны позже в данной главе.

Второй частью схемы работы с контентом является процесс перевода записей из состояния черновика в состояние завершенной публикации, по возможности с более подробным рассмотрением статусов «Личное», «Запланированная» и «На утверждении». Запись, сделанная участником, будет находиться в статусе «На утверждении» до момента ее публикации с чьего-то разрешения. Большая часть рабочего процесса с записями связана с закладкой Записи (Posts) в Консоли, где четко помечается статус каждой записи, а также присутствуют прочие пункты меню, разрешающие публикацию или производящие иные изменения в статусе записи, такие как пометка «Личное» или задание времени ее публикации в будущем.

Если у вас на сайте WordPress пишут нескольких авторов, не забывайте о том, что все его содержимое хранится в единой базе данных MySQL, это позволяет пользователям легко видеть текущее состояние контента. WordPress содержит функцию `wp_transition_post_status()` для плагинов, которые пытаются отслеживать изменения в статусе отдельной записи или же обновляют динамическую страницу, а также иным способом уведомляют пользователей о внесении изменений в схему работы.

Одним из особенно интересных для рассмотрения плагинов является Edit Flow (http://wordpress.org/extend/plugins/edit-flow/). Данный плагин был разработан для моделирования схемы публикации новостей в газете через WordPress. Он включает несколько статусов для новых записей, начиная с подачи заявки на публикацию и заканчивая традиционной публикацией записи. Он также содержит особые календари обеспечения историй, позволяющие гарантировать надлежащее распределение вашего контента и его передачу писателям и фотографам для соответствующего завершения. Кроме того, данный плагин включает скрытую переписку между редактором сайта и автором, что упрощает схему работы и публикацию записей удаленных авторов. Это абсолютно нишевый плагин, но если вам требуются такие возможности, то обратите на него внимание.

Организация контента

Ведутся постоянные споры на грани объявления войны об использовании записей на сайтах WordPress, которые используются в качестве стандартных сайтов со статичным контентом. Записи являются неотъемлемой частью WordPress, они абсолютно применимы и полезны на любом сайте. Однако основной причиной дискуссий является то, что записи обычно ведутся в хронологическом порядке, и только содержимое, организованное по времени, вписывается в данную парадигму. Записи могут быть представлены в форме небольших блоков контента, используемых в самых разных целях, а одна из функций WordPress в качестве CMS связана с изменением стратегии мышления относительно того, какие типы контента используются и какое влияние они имеют на сайт.

Ниже приведены три простых примера использования записей на коммерческом сайте:

1. Создайте запись для каждого продаваемого товара. Комментарии к записи позволяют пользователям выразить свое отношение и предложить рекомендации.
2. Создайте рубрику или метку для каждого товара на сайте, а затем организуйте записи о товаре. Первой записью в каждой рубрике должна быть информация о товаре и, возможно, ссылка на корзину, с помощью которой товар можно приобрести. Теперь вы можете использовать структуру этих записей для предоставления более подробной информации о каждом товаре: почему вы его предлагаете, как он создается, определяется или распределяется, какие о нем существуют еще отзывы, обзоры или комментарии.

3. При создании раздела помощи по конкретному товару для каждой вспомогательной темы может быть отдельная запись. Каждая вспомогательная запись может содержать один небольшой блок контента, касающийся конкретной задачи или характеристики, и использовать ярлыки или механизмы категоризации для сортировки и обеспечения навигационной информацией пользователей, пытающихся помочь себе самостоятельно. Комментарии к записям позволяют пользователям описывать относительную полезность каждой записи. Аналогичным образом пусть и не связанная с электронной коммерцией команда jQuery использует этот метод для регистрации API записи на http://api.jquery.com, одновременно создавая за счет комментариев как ресурс поддержки, так и ресурс сообщества.

В каждом из этих простых примеров вы можете переключить заданное по умолчанию поведение WordPress с показа наиболее свежих записей и вместо этого создать смесь статичного и динамического контента домашней страницы, ассоциирующейся со статичным сайтом. Вы можете найти множество примеров применения систем управления контентом WordPress на http://wordpress.org/showcase/tag/cms.

Поддержка тем и виджетов

Поддержка тем является ключевой в управлении контентом. Вашей конечной целью может быть не изменение внешнего вида WordPress, так чтобы он был уж точно не похож на блог, а простой поиск темы, которая предоставляет достаточную гибкость для отображения каждого типа контента в подходящем стиле вне зависимости от того, относится он к сетевым продажам вашей продукции или связан с рассылкой новостей. Здесь рассмотрен весьма полезный случай использования темы Thematic (http://wordpress.org/extend/themes/thematic) с ее тринадцатью областями виджетов и широкими возможностями расширения для дочерних тем. Как и в самом крайнем случае, тема P2 (http://p2theme.com), разработанная Automattic, размещает панель записей, обновления в реальном времени и возможность редактирования прямо на домашней странице, что позволяет сочетать Twitter, блог, форум для обсуждения и новостной сайт. С помощью тем ваш сайт может полностью измениться и стать непохожим на любой другой сайт WordPress.

Если вы собираетесь использовать тему с областями виджетов и хотите расширить количество типов контента, доступных в этих боковых панелях, вам может понравиться TinyMCE — редактор на основе JavaScript для превращения областей HTML-текста в нечто более подходящее для выбранной темы. Плагин Rich Text Widgets подходит для решения проблемы с заданными по умолчанию текстовыми виджетами, поддерживающими только простой текст. Когда вы активируете Rich Text Widget, написанный Жульен Апперт (http://wordpress.org/extend/plugins/rich-text-widget/), как показано на рис. 14.2, вам становится доступен новый виджет на панели управления виджетами. Этот виджет содержит встроенный редактор TinyMCE, что позволяет вашим редакторам размещать на этой боковой панели нечто большее, чем простой текст.

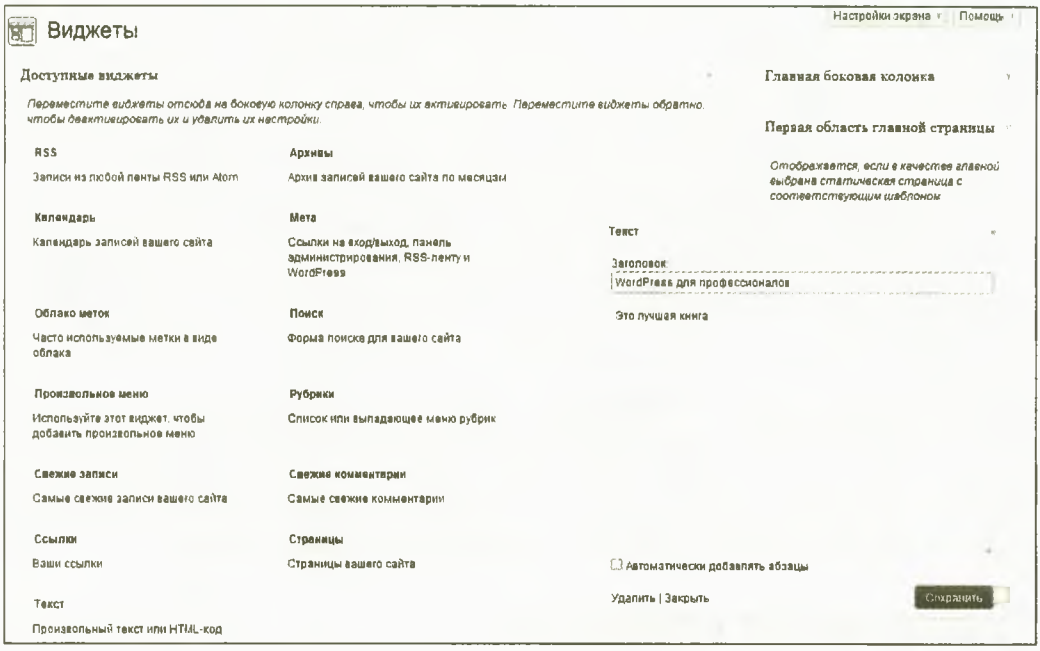


Рис. 14.2. Редактирование виджета Rich Text

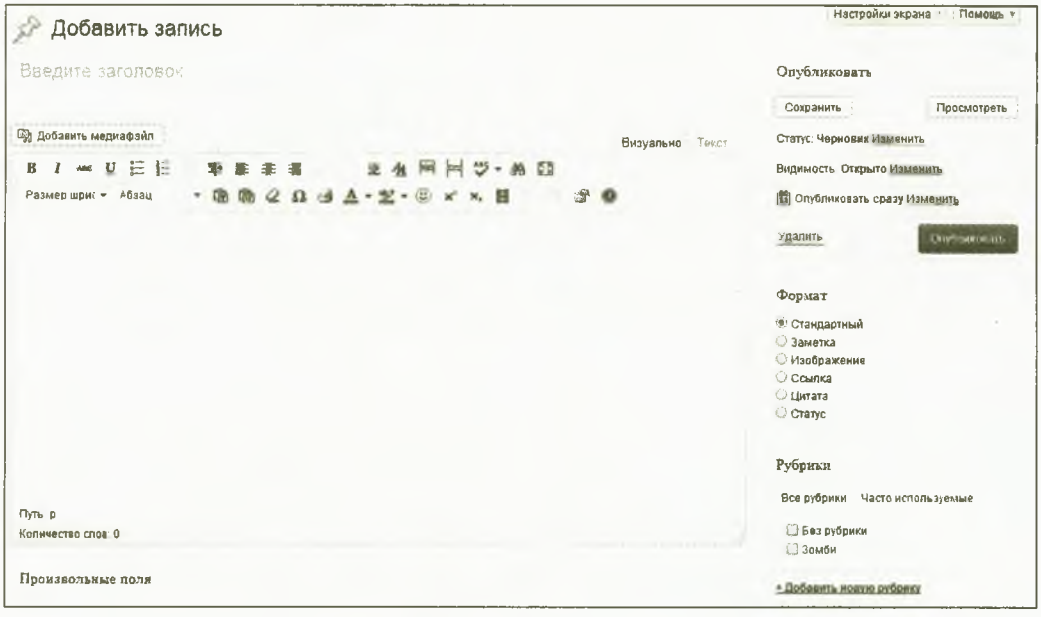


Рис. 14.3. Использование TinyMCE Advanced для редактирования записи

При использовании данного плагина вам необходимо работать с некоторой осторожностью. Обычно ваша боковая панель, как и другие области с готовыми виджетами, имеет фиксированную ширину. При использовании описанного плагина ваш разработчик контента может загрузить в область виджета все, что ему угодно, а это может нарушить разметку сайта. Тем не менее эта проблема будет решена после некоторой тренировки.

Встроенный редактор TinyMCE вполне работоспособен, хотя и не поддерживает некоторые часто используемые возможности, такие как редактирование таблиц. Плагин TinyMCE Advanced Эндрю Озза (http://wordpress.org/extend/plugins/tinymce-advanced/) дополняет вышеупомянутый плагин и позволяет компенсировать его недостатки. Тем не менее стоит помнить о том, что чем сложнее ваш контент, тем больше риск разрушить ваш сайт неправильно отформатированными или скомпонованными таблицами. На рис. 14.3 вы можете увидеть некоторые возможности плагина в действии, хотя их предлагается намного больше.

Домашние страницы

Помните уроки писательского мастерства в старших классах, где в каждой истории должна была быть сюжетная изюминка? Изюминка появлялась в тот момент, когда вам удавалось сделать свой рассказ уникальным, интересным и захватывающим. Подходящая изюминка для сайта на WordPress также позволит захватить читателя и извлечь пользу из оставшейся части вашей истории, будь то контент, относящийся к конкретному товару, или смесь из статичных и зависящих от времени записей.

Общепринятый подход подразумевает создание статичной главной страницы. Как описано в главе 9, вы можете сделать это несколькими различными способами. Самым простым из них является использование панели инструментов WordPress Reading Settings (Настройки чтения), чтобы задать статичную главную страницу для вашей домашней страницы. Кроме того, на этой странице может быть использован шаблон для изменения и разграничения разметки остальной части вашего сайта. Вам понадобится создать страницу (а не запись) именно для этой цели, а затем использовать Консоль для задания ее в качестве главной страницы.

Альтернативой такому подходу является использование особого файлового шаблона WordPress в качестве вашей главной страницы, заменяющей список записей по умолчанию. В вашей теме WordPress найдите файловый шаблон под названием `front-page.php`, как это описано в главе 9. С помощью файлового шаблона вы добьетесь большей гибкости в раскладке и функционале вашей стартовой страницы, так как сможете напрямую редактировать PHP-код, выбирая, например, размещать первыми приклеенные записи или же рекламируемые товары, а затем уже связанный с ними контент или наиболее свежие записи. Используя комбинацию информационных полей дополнительных страниц и произвольных циклов, описанных в главе 5, вы сможете индивидуализировать подборку содержимого вашей домашней страницы настолько детально, насколько захотите.

Страницы избранных объектов

Отличным инструментом для создания изюминки вашей статичной домашней страницы является избранный объект. Такая уловка весьма популярна среди тем в журнальном стиле, а также на многих других сайтах. Очень часто в верхней трети области контента вы видите большую область с изображением и заголовком, выделяющим элемент из всего остального содержимого сайта. Такое расположение называется «hero spot» (софит) и часто используется на популярных сайтах. Это связано с его эффективностью.

В целом при внедрении избранного объекта на стартовую страницу вам хочется иметь в распоряжении несколько различных изображений и использовать jQuery (или другую библиотеку JavaScript) для их пролистывания. Таким образом, вы не рассчитываете полностью на единственный «софит», но делаете акцент на нескольких различных идеях, и если повезет, одна из них привлечет внимание посетителя сайта. Ваша цель — выделить объекты, управляющиеся как записи, и настроить типы записей или аудиовизуальную информацию с помощью WordPress так, чтобы редакторы могли управлять ими в большей степени, чем администратор сайта.

В первую очередь необходимо настроить систему под избранные объекты. Вы можете использовать рубрику «Features» (особенные товары) и вывести в слайдшоу только записи из нее. Однако это означает, что в других частях сайта, если там присутствует раздел новостей, где выводятся все записи, вам придется исключить данную категорию из циклов, которые могут причинить неудобства. Так можно использовать эту возможность для сайта на WordPress, и в некоторых темах функция до сих пор реализована именно таким образом.

Тем не менее, как указано в главах 7 и 9, вы можете использовать настройки типов записей для реализации той же функции без дополнительных затрат на выполнение циклов. Опять же, это всего лишь один из способов реализации такого типа управления контентом; существует множество других. В данном примере вы будете напрямую редактировать файловые шаблоны, хотя описанную функцию также можно встроить в плагин.

Допустим, задача состоит в том, чтобы продемонстрировать три случайных товара в качестве слайдов. Сперва зарегистрируйте новый тип записи «слайды», как это описано в главе 7. В файле `functions.php` содержится необходимый код. В нашем случае он будет выглядеть примерно так:

```
/*
 * Слайды для особенных продуктов
 * Регистрируем тип записи для продуктов
 */
add_action( 'init', 'wppro_create_post_types' );
function wppro_create_post_types() {
    register_post_type( 'slides',
        array(
            'labels' => array(
```

```
            'name' => _x( 'Slides', 'post type general name' ),
            'singular_name' => _x( 'Slide', 'post type singular name' ),
            'add_new' => _x( 'Add New', 'Slide' ),
            'add_new_item' => __( 'Add New Slide' ),
            'edit_item' => __( 'Edit Slide' ),
            'new_item' => __( 'New Slide' ),
            'view_item' => __( 'View Slide' ),
            'search_items' => __( 'Search Slides' ),
            'not_found' => __( 'No Slides found' ),
            'not_found_in_trash' => __( 'No Slides found in Trash' ),
            'parent_item_colon' => ''
        ),
        'public' => true,
        'exclude_from_search' => true,
        'supports' => array('title','thumbnail','editor'),
        )
    );
}
```

Для отображения на вашем сайте вам необходимо либо отредактировать файловый шаблон `front-page.php`, либо создать новый файловый шаблон для вашей стартовой страницы. Приведенные изменения в коде должны быть включены в этот файл.

Далее вам нужно извлечь слайды из базы данных. Вы можете получить эти записи с помощью простой функции WordPress `get_posts()`:

```
global $post;
    $args = array(
    'post_type' =>'slides',
    'numberposts' => 3
    'orderby' => 'ASC'
);
$slider_posts = get_posts($args);
```

По заданным параметрам вы можете понять, что функция возвращает три заданные записи в восходящем порядке. Вы можете изменить `numberposts` на –1, что позволит вам получать только одну из записей. Далее в области содержимого файлового шаблона вам необходимо добавить немного HTML для jQuery и использовать цикл на результирующем объекте:

```
<div id="slideshow_container">
    <div id="slideshow">
        <?php if($slider_posts) {
            foreach($slider_posts as $post) : setup_postdata($post);
                // извлекаем изображение
                $thumbnail = wp_get_attachment_image_src(get_post_thumbnail_id(),
                    'home-slide');
                if ($thumbnail[1] == "600" && $thumbnail[2] == "160") {
                // проверяем размеры миниатюры в css ?>
                <div id="feature-<?php echo $post->ID; ?>" class="slide">
                <a href="<?php the_permalink(); ?>" title="<?php the_title(); ?>">
                <img src="<?php echo $thumbnail[0]; ?>" title="<?php the_title();
                ?>" />
                </a>
```

```
                </div>
                <?php } ?>
            <?php endforeach; ?>
            <?php wp_reset_postdata();
        } ?>
    </div>
</div>
```

Суть состоит в том, что вы создаете некоторые `divs` оболочки для слайд-шоу, зацикленные вокруг каждого заданного типа записи в результатах запроса и выводящие избранное изображение. При использовании описанной последовательности избранное изображение будет иметь заданные размеры, которые необходимо проверить, чтобы изображение не нарушало разметку сайта. И наконец, избранное изображение становится ссылкой, связанной с содержимым записи, созданной вами через консоль.

Это может оказаться чрезвычайно практичным, потому что вы настроили рекламные слайды на вашей стартовой странице как ссылки на отдельные целевые страницы, что делает удобным отслеживание рейтингов при анализе трафика. Более того, вы смогли связать традиционные записи и содержимое страниц с абстрактными рекламными записями.

Если вы посмотрите на сайт теперь, то увидите несколько наслаивающихся изображений, соответственно, следующим шагом будет их разделение с помощью небольшого волшебства JavaScript и превращение их в слайд-шоу. Многие являются большими поклонниками плагина jQuery Cycle Майка Алсупа (http://malsup.com/jquery/cycle/). Обратите внимание на то, что это — плагин для jQuery, а не для WordPress. Данный плагин очень прост в использовании и включает несколько удобных возможностей перехода. С помощью этого плагина вы можете конвертировать HTML в слайд-шоу с автоматической прокруткой. Приведенный ниже код JavaScript обращается к «подвалу» вашего файлового шаблона или к той части, где вы обычно размещаете JavaScript. jQuery может выглядеть примерно так:

```
$('#slideshow').cycle({
    fx:     'scrollHorz',
    speed:   500,     // время, в течение которого длится переход
    timeout:  10000,  // время между переходами
    pause:   1,       // остановить переход по наведению мыши
    random:  0,       // произвольный порядок
    delay:   -1000,   // задержка перед началом первого перехода
    next:   '#next',
    prev:   '#previous',
    pager:   "#slidenav",
    pagerEvent:'mouseover',
    // название события, которое управляет пейджингом
    autostop: true,
    // true останавливает слайд-шоу после X переходов
    // (где X == число слайдов, используйте эту настройку,
    // чтобы обеспечить работу в браузерах без javascript)
    autostopCount: 100,
    // число переходов (может быть использовано для значения
```

```
    // autostop, чтобы задать X)
    // вызов функции, которая создает миниатюру, чтобы
    // использовать ее в качестве якоря ссылки
    pagerAnchorBuilder: function(idx, slide) {
        slide.id = "slide"+idx;
        var desc = jQuery('#'+slide.id+' img:first').attr("title");
        return '<a href="#" title="'+desc+'">'+(1+idx)+'</a>';
    }
});
```

Экспериментируя с различными эффектами и настройками времени, а также используя другие параметры, вы можете добиваться уникальных результатов. Этот же метод может быть использован для демонстрации некоторых отзывов. Их можно оставить в виде слайд-шоу, как было указано выше, или же выводить случайную запись из рубрики отзывов. Вместо использования изображений в качестве контента вы можете работать с фактическими отформатированными данными записей. Вы ограничены лишь пределами вашего воображения.

Иерархия контента

Помимо избранных объектов для привлечения внимания пользователя большинство систем управления контентом позволяют вам создавать иерархию содержимого, облегчающую навигацию, уже после того, как вы завладели вниманием пользователя. Стандартная иерархия содержит контент различных типов в структуре дерева для внедрения навигационных схем, позволяющих комбинировать свойства статичного контента со свойствами более динамического.

Одним из очевидных путей создания иерархии контента является использование рубрик и меток для сортировки записей по соответствующим группам. Это весьма полезно при использовании записей как основного типа контента и организации их в эквивалентные классы по рубрикам или в соответствии с данными уникальных меток. Помимо этого, из предыдущего раздела данной главы вы узнали, как можно использовать записи для создания избранного объекта или слайд-шоу на стартовой странице.

Пользовательские таксономии, рассматриваемые в главе 7, содержат еще один способ организации и поиска записей. Он предоставляет вам еще большую гибкость при настройке цикла темы. Данный раздел рассматривает пользовательские страницы и иерархии записей более подробно, приводя примеры плагинов, позволяющих вам работать с аспектами управления контентом в WordPress, подстраивая их под желаемый стиль вашего сайта.

Вы можете начать с проблемы сложной навигации. Например, у вас есть гибридный сайт с разделами, где страницы имеют очевидные типы контента, но также и с разделами, в которых использование рубрик записей будет иметь больше смысла. Вы не хотите работать со сложным по коду навигационным деревом, а использование меню также не решает проблему, потому что содержание записей (намеренно) регулярно изменяется.

Одним из решений является плагин Марка Джакита — Page Links To (http://txfx.net/wordpress-plugins/page-links-to/). Этот плагин создает новое поле на панели Написать (Write) вашей страницы. Используя это поле, вы можете создавать страницы, которые ссылаются на другие страницы. Это позволяет вам применять систему меню в качестве общей системы навигации по сайту и при этом создавать пункты меню, которые направляют пользователя по ссылкам за пределы сайта. Или, что встречается более часто, позволяют вам создать пункт меню как страницу, которая направляет пользователя на страницу цикла записи.

Например, предположим, что в меню О нас (About Us) у вас выложены традиционная История компании (History) и страницы Контакты (Contact Us), но помимо этого имеется страница с объявлениями о приеме на работу. И ваша компания использует услуги третьей стороны для публикации объявлений о приеме на работу, что создает соответствующую проблему в навигации. С помощью описанного плагина вы можете создать новую страницу под названием Вакансии (Careers), которая, в свою очередь, будет ссылкой на сайт третьей стороны с объявлениями. Вы можете увидеть, как это делается, на рис. 14.4.

Рис. 14.4. Создание страницы, ссылающейся на сайт третьей стороны с помощью плагина Page Links To

Еще одной проблемой большого сайта является обслуживание страниц. С течением времени ваш сайт разрастается до появления множества подстраниц в рамках

исходных страниц. Данная структура необходима для облегчения восприятия вашего сайта, что описано в главе 9. Темы включают несколько возможностей для управления общей навигацией, и очень важно определить, какой именно метод использован в вашей теме. Большинство тем используют систему меню WordPress.

Система меню является удобной и гибкой, но она отделена от фактического контента. С помощью системы меню вы можете автоматически добавлять страницы в систему навигации, но только на самом высоком уровне. Однако если вы удаляете страницу с содержимым из панели инструментов, это действие не влияет на меню, что оставляет вас с пустой навигационной ссылкой. Такой отрыв может быть весьма пугающим для начинающих администраторов WordPress и может раздражать более опытных администраторов невозможностью связать две системы.

Второй, более старый метод использует встроенный функционал исходных страниц. Управление страницами начинает работать, но может оказаться неожиданно быстрым в случае наличия большого количества страниц. Когда же вам необходимо переместить страницы, применение этого метода оказывается довольно утомительным, что вы можете видеть на рис. 14.5.

Рис. 14.5. Выпадающее меню исходной страницы на сайте с небольшим количеством страниц

Один из способов простого управления меню с помощью ассоциирования контента с объектом навигации встроен в WordPress, это плагин Page re-Mash, написанный Джоелом Старнсом и обновленный Мэттом Макинвелом. Для того чтобы плагин Page re-Mash работал с текущей версией WordPress, загрузите оригинальный PageMash с http://wordpress.org/extend/plugins/pagemash/, а затем замените файл `pagemash.php` его обновленной версией с http://binarym.com/2010/pagemash-trash-auto-draft-and-wordpress-3-0/.

Page re-Mash за счет своих AJAX- возможностей позволяет вам перемещать страницы, просто перетаскивая их. Как вы видите на рис. 14.6, страницы могут быть размещены и организованы в интуитивно понятный интерфейс. Page re-Mash требует, чтобы в вашей теме использовалась функция `wp_list_pages()` для навигации по сайту, а не встроенная система меню WordPress. Данный плагин позволит вам

скрыть страницы от функции `wp_list_pages()`, предоставляя даже больше контроля над полной структурой навигации по сайту.

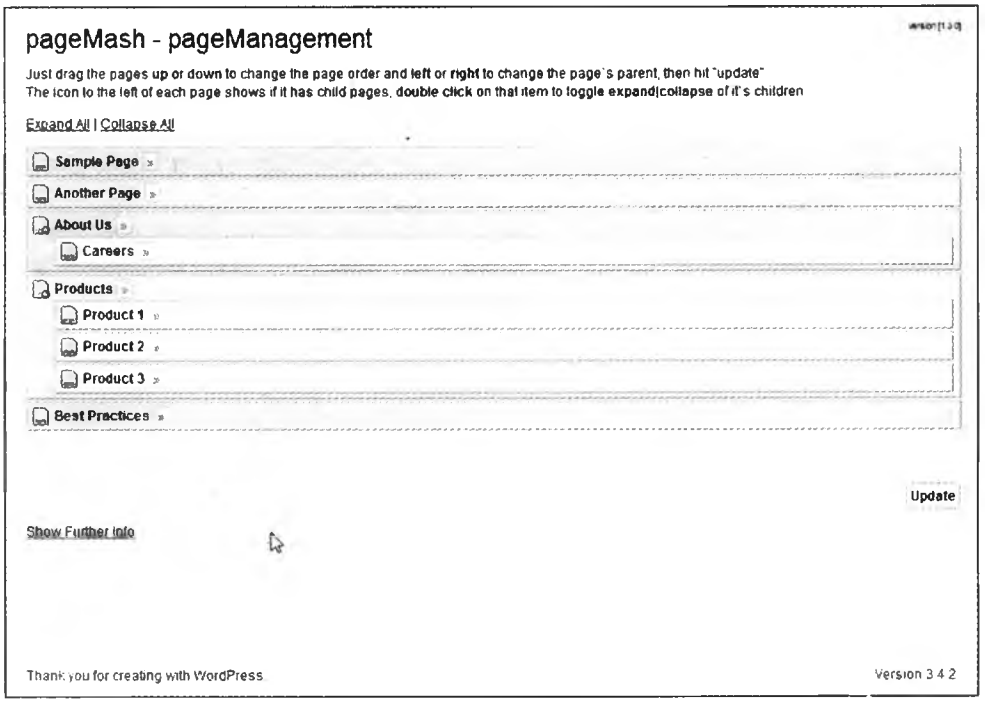


Рис. 14.6. Управление порядком и размещением страниц с помощью плагина PageMash

Предположим, вы следуете концепции: «записи являются универсальным контентом» и используете на сайте записи с различными типами содержимого, тогда очень полезным дополнением для вас послужит плагин Майкла Йошитаки Эрленвайна — Yet Another Related Post (http://wordpress.org/extend/plugins/yet-another-related-posts-plugin/). Данный плагин предлагает вам список дополнительного контента, который потенциально имеет отношение к текущей записи. Использование этой функции будет стимулировать посетителей исследовать другие, сходные по тематике области сайта. Иными словами, данный плагин создает перекрестные ссылки между связанными между собой элементами содержимого вашего сайта.

Плагин Pods Framework (http://wordpress.org/extend/plugins/pods/) позволяет зайти в структурировании данных еще дальше. Pods дает возможность создавать новые типы контента над и под записями и страницами, а также настраивать произвольные поля. Вы можете стандартизировать эти новые типы за счет отношений между данными. Вы также можете использовать эти новые типы контента и отношения на страницах Pods, которые поддерживают особый синтаксис Pods и PHP-код. Pods является намного более сложной системой, но и намного более мощной.

Интерактивные свойства

Самым главным интерактивным свойством является поиск, рассматриваемый в главе 12. Другие базовые механизмы взаимодействия с пользователем, ассоциируемые с CMS, — это форумы, формы и базовые возможности электронной коммерции.

Форумы

Комментарии к записям — самый простой вид содержимого обсуждения. Время от времени вам может захотеться отойти от контента, который вы создаете, в попытке перевести общение в инициируемый пользователем диалог. Форум представляет собой открытое обсуждение, во времена, предшествовавшие созданию широкополосного Интернета, обычно называемое доской объявлений. Самый простой способ добавления форумов предлагается проектом bbPress (http://bbpress.org), связанным с WordPress и также поставляемым Automattic. bbPress будет делиться данными пользователей с WordPress, соответственно зарегистрированные пользователи смогут участвовать в форумах и обсуждениях. Имеется возможность загружать и те и другие, после чего форумы будут отображаться как раздел вашего сайта.

Если вам необходимы более простые функции форума как вида контента, то желаемый результат, вероятно, даст просмотр доступных плагинов. С другой стороны, если вы хотите интегрировать многочисленные хранилища для управления контентом в единый пользовательский опыт, возьмите страницу (в прямом смысле) из WordPress.org: bbPress будет служить источником для созданной пользователями области поддержки на WordPress.org.

Формы

Еще одной проблемой сайта является создание и управление формами, такими как формы связи и другие аналогичные формы вплоть до электронной переписки. Существует огромное количество плагинов для форм связи, но для полноценного сайта вам, скорее всего, придется выйти за их пределы. В течение очень долгого времени плагин cForms II Оливера Зейделя (http://www.deliciousdays.com/cforms-plugin) был всеобщим фаворитом. cforms работает очень хорошо и обладает исключительной мощностью. Вы можете настраивать его любым желаемым способом, но пользовательский интерфейс, увы, не очень хорош. Вы просто не сможете передать cforms вашему контент- администратору, чтобы он создавал новые формы.

Альтернативой CForms является Gravity Forms, созданный RocketGenius (http://www.gravityforms.com/). Это один из немногих плагинов, упомянутых в данной книге, использование которого является платным. Но, честно говоря, Gravity Forms — весьма интригующий проект. Спросите любого разработчика WordPress о его любимом плагине, и вы узнаете, что он использует Gravity Forms. Gravity Forms позволяет вам создавать любой необходимый тип формы через очень простой, напоминающий AJAX интерфейс. Это настолько просто, что большинство контент-администраторов

могут показать, как плагин работает. Не слишком мощный, но с четким интуитивно понятным интерфейсом, он действительно один из лучших среди плагинов.

Кроме того, с точки зрения пользовательского интерфейса визуализация HTML выполнена на высочайшем уровне. Он выглядит просто фантастически и не требует никакого дополнительного изменения стиля. Однако если у вас возникнет желание изменить его, то для вашего удобства HTML наполнен механизмами привязки идентификационных данных CSS-класса. Кроме того, этот плагин настолько популярен, что многие темы включают стили для него. Последним мощным свойством Gravity Forms является то, что помимо стандартной функции «отправки формы по указанному адресу» он также имеет модуль для Консоли, с помощью которого можно отследить отправленные формы. Несмотря на то что CForms имеет аналогичные функции, полезность Gravity Forms оправдывает его стоимость.

Электронная коммерция

Если вы разрабатываете сайт с избранными товарами и страницами, метками товаров и их рубриками, вы, скорее всего, собираетесь что-то продавать. Интеграция покупательской корзины и системы оплаты являются еще одной, последней, категорией во взаимодействии с пользователем. Если вы поищете в директории с плагинами на WordPress.org, вы наткнетесь по меньшей мере на дюжину различных плагинов для создания покупательской корзины и проведения оплаты. Вместо того чтобы перечислять их, мы приведем краткий список того, на что стоит обратить внимание:

- Насколько сложна настройка покупательской корзины? Собираетесь ли вы обременять своих администраторов мелкими делами, такими как обновление перечня товаров со скидкой, или же вы можете передать эту задачу контент-менеджерам?
- Какую статистику по покупательским корзинам вы хотите вести? Ознакомление с тем, как и почему пользователи отказываются от товаров, поможет вам улучшить свой сайт либо за счет предоставления большей информации о товаре, либо с помощью упрощения процедуры покупки.
- Какие платежные системы поддерживаются? Если вас не интересует нечто более сложное, чем возможность принимать платежи через PayPal после введения номера конкретного товара или объекта, вы можете воспользоваться шаблонами кнопки PayPal и вручную интегрировать ее в боковые панели или ваши страницы.

Плагин WP e-Commerce (http://getshopped.org) сочетает в себе практически все описанные механизмы расширения функциональности: использует настраиваемые типы записей для страниц товаров, организует их в настраиваемые иерархии и позволяет добавлять специфические для плагина таблицы баз данных для управления характеристиками товара. Он интегрируется с множеством механизмов оплаты и четко демонстрирует, какого стиля можно ожидать от собственной сборки в WordPress.

Другие системы управления контентом

Вы уже ознакомились с тем, насколько WordPress больше, чем простая платформа для блогов. Он может быть использован для создания разнообразных сайтов, но помимо того, что достигается за счет плагинов, он предоставляет пользователям возможности ядра WordPress. Иногда вам хочется дополнить свой сайт функциями, которые можно обнаружить в традиционных сетевых приложениях, таких как форумы, подобные bbPress, социальные сети, такие как BuddyPress, приложения для электронной коммерции и решения из других CMS.

С учетом широчайшего выбора доступных систем управления содержимым у большинства компаний уже есть одно, а то и несколько своих хранилищ контента, которые прекрасно работают, и часто необходимо интегрировать WordPress с другой CMS. Данный раздел вкратце рассматривает вопрос, когда стоит использовать WordPress в качестве внешнего потребителя или производителя контента, а когда не стоит обращаться к WordPress как к главной системе управления им.

Интеграция WordPress

Интеграция такого приложения, как WordPress, с другим ориентированным на содержимое приложением требует от вас подгонки их управления пользователями, архивирования содержимого и, возможно, аспектов стиля друг под друга. Эта задача проще всего решается с помощью плагинов или тем и обычно требует связующего кода. Основу для него можно найти в директории плагинов, а в данном разделе представлен краткий обзор наиболее популярных вопросов.

Как внешняя система передает контент? Получаете ли вы ленту RSS и в каком случае вы используете агрегатор RSS, рассматриваемый в главе 11, в качестве отправной точки, или же вы получаете сырой JSON, требующий редактирования и интерпретации перед превращением его в запись?

Если у вас есть удаленный источник URL, то можете ли вы встроить его с помощью плагина от провайдера oEmbed (http://oembed.com)? oEmbed получает электронный адрес и возвращает контент различных типов, который может быть интегрирован в темы WordPress, что позволяет отображать его, не заставляя WordPress интерпретировать вид контента.

Нужно ли вам управлять авторизационными данными пользователя между сайтами? Необходимо ли хранить информацию о входах пользователя на внешний сайт (в базе данных WordPress MySQL с помощью таблицы, создаваемой вашим плагином) и, если необходимо, работаете ли вы с условиями возникновения ошибок, такими как смена пароля или удаление пользователя в удаленной системе?

Можно ли воспринимать WordPress исключительно в качестве движка для блогов, производящего записи для таких систем управления контентом, как Drupal? В таком случае WordPress становится частью Drupal, управляющей ее контентом

в качестве источника записей для блогов, но передающей контроль над их презентацией настройкам Drupal.

Где не стоит использовать WordPress

Не всякая проблема в управлении контентом может быть решена с помощью WordPress. Иногда вам может пригодиться и другой инструмент или даже набор инструментов для решения задачи:

- **Работа с мультимедийной рекламой.** Потоковое видео, аудио и изображения с огромным количеством метаданных могут быть отображены или включены в записи WordPress, но если вам необходимо создавать метки к классификационным видеофайлам или производить поиск изображений на основе их EXIF-меток, вам, вероятно, понадобится CMS, разработанная для управления мультимедийной рекламой.
- **Процессор для сложных интернет-приложений.** Грядет новое поколение мобильных клиентов. Обычно они связаны с обсуждением постпроцессорного хранилища данных, а не полноценного сайта. WordPress же генерирует HTML, а не JSON или другой формат архивирования данных, воспринимаемый API. WordPress может выполнять данную функцию, но вот является ли он идеальным инструментом?
- **Простой сетевой магазин.** Если вы просто планируете создать магазин, воспользуйтесь конструктором магазинов, оснащенным покупательской корзиной, системой оплаты и каталогом продукции. Вам, возможно, будет не хватать интеграции обсуждения товара, обратной связи, рекомендаций или возможности рассказать о том, почему вы представляете (или создали) конкретный товар. И хотя все описанное реализуемо с помощью возможностей, приведенных в данной главе, иногда вам просто необходимо, чтобы покупатели имели возможность приобрести товар за несколько щелчков мыши.
- **Календари событий.** Несколько сайтов управляют календарями, ведут учет событий, размещают информацию о них, предлагают уведомления и напоминания о приближающихся событиях из календаря. WordPress все еще не имеет встроенного достаточно функционального календаря и плагинов для управления событиями, что делает календари областью, в которой использование Drupal или BuddyPress будет более простым и проработанным. BuddyPress в особенности имеет большое количество возможностей для работы и размещения контента по календарю, хотя и предлагает вам скорее инструмент управления сообществами, напоминающий частную социальную сеть, чем систему управления контентом в чистом виде.
- **Нестандартные решения.** Если вам необходимо модифицировать или вносить изменения в ядро WordPress, то либо вы в чем-то ошибаетесь, либо — что более вероятно — WordPress — не то решение, которое вам нужно. При взломе ядра кода WordPress вы изменяете путь для установки обновлений так, чтобы иметь

возможность переписать любые изменения, внесенные в пакет файлов ядра, и заставить вашу функцию работать, как только будет выпущена новая версия WordPress. Как было сказано в главе 4, не взламывайте ядро.

- **Перебор плагинов.** Плагины правят миром, и большая часть данной книги была посвящена выбору подходящих плагинов для конкретных функций или же пояснению того, как создавать для тех же функций собственные. Но вы можете перейти границы. Если вы будете использовать слишком много плагинов, вы можете снизить эффективность работы сайта или сделать его более уязвимым в результате неизвестных взаимосвязей в работе плагинов. Каждый плагин также увеличивает технические требования к ресурсам для сайта, особенно к памяти и рабочей мощности.

Резюме

WordPress представляет собой мощную систему управления контентом, обладающую множеством свойств, присущих коммерческим системам, предшествовавшим блог-буму 2000-х. В равной степени благодаря своим корням с открытым исходным кодом, сильной команде разработчиков и гибкому расширяемому дизайну WordPress зарекомендовал себя как инструмент, во многом превосходящий простой движок для блогов.

Б. Уильямс, Д. Дэмстра, Х. Стэрн

WordPress для профессионалов

Серия «Для профессионалов»

Перевела с английского *Александра Вареникова*

Заведующий редакцией	*А. Юрченко*
Ведущий редактор	*Ю. Сергиенко*
Научный редактор	*А. Гребеньков*
Литературный редактор	*И. Васильева*
Художник	*Л. Адуевская*
Корректоры	*С. Беляева, Н. Викторова*
Верстка	*Л. Егорова*

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Подписано в печать 17.04.14. Формат 70×100/16. Усл. п. л. 37,410. Тираж 1500. Заказ № 164.

Отпечатано в полном соответствии с качеством предоставленных издательством материалов
в ГПО «Псковская областная типография». 180004, Псков, ул. Ротная, 34.

ИЗДАТЕЛЬСКИЙ ДОМ
ПИТЕР®
WWW.PITER.COM

ПРЕДСТАВИТЕЛЬСТВА ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
предлагают профессиональную и популярную литературу по различным направлениям: история и публицистика, экономика и финансы, менеджмент и маркетинг, компьютерные технологии, медицина и психология.

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-73, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: voronej@piter.com

Екатеринбург: ул. Бебеля, д. 11а
тел./факс: (343) 378-98-41, 378-98-42; e-mail: office@ekat.piter.com

Нижний Новгород: тел.: 8 960 187-85-50; e-mail: nnovgorod@piter.com

Новосибирск: Комбинатский пер., д. 3
тел./факс: (383) 279-73-92; e-mail: sib@nsk.piter.com

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 229-68-09; e-mail: samara@piter.com

УКРАИНА

Киев: Московский пр., д. 6, корп. 1, офис 33
тел./факс: (044) 490-35-69, 490-35-68; e-mail: office@kiev.piter.com

Харьков: ул. Суздальские ряды, д. 12, офис 10
тел./факс: (057) 7584145, +38 067 545-55-64; e-mail: piter@kharkov.piter.com

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163
тел./факс: (517) 208-80-01, 208-81-25; e-mail: minsk@piter.com

---

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок
тел./факс: (812) 703-73-73; e-mail: spb@piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов
тел./факс издательства: (812) 703-73-72, (495) 974-34-50

Заказ книг для вузов и библиотек
тел./факс: (812) 703-73-73, доб. 6250; e-mail: uchebnik@piter.com

---

Заказ книг по почте: на сайте www.piter.com; по тел.: (812) 703-73-74, доб. 6225

Эта книга, выходящая во втором издании, поможет вам стать экспертом в разработке сайтов на базе платформы WordPress. WordPress является самой популярной в мире бесплатной CMS-системой, однако большинство разработчиков используют только базовые функции WordPress, не углубляясь в профессиональную веб-разработку на ее основе. Вместе с тем если использовать WordPress по максимуму, на его базе можно создавать проекты любого уровня сложности и дизайна.

В книге подробно описана система CMS, ее основные функциональные элементы, внутренняя работа кода и структуры данных. Рассказывается о разработке собственных дизайн-тем, использовании плагинов и написании собственных расширений, настройке и оптимизации крупных ресурсов, работающих на этой системе. Издание содержит большое количество примеров и готового кода, который можно использовать в своих проектах.

Книга адресована широкому кругу разработчиков: от тех, кто стремится выполнить тонкую настройку темы WordPress, до более опытных разработчиков, знакомых с разработкой плагинов.

Тема: веб-разработка на WordPress | Уровень пользователя: опытный

ISBN: 978-5-496-00948-5

9 785496 009485

ПИТЕР

Заказ книг:
Санкт-Петербург
тел.: (812) 703-73-74, postbook@piter.com
www.piter.com — каталог книг и интернет-магазин