

O'REILLY®

2-е издание

Разработка веб-приложений на WordPress

WordPress как фреймворк



Брайан Мессенленер
Джейсон Коулман



SECOND EDITION

Building Web Apps with WordPress

WordPress as an Application Framework

Brian Messenlehner and Jason Coleman

Брайан Мессенленер
Джейсон Коулман

Разработка веб-приложений на WordPress

WordPress как фреймворк

Санкт-Петербург
«БХВ-Петербург»

2021

УДК 004.4'236
ББК 32.973.26-018
М53

Мессенленер, Б.

М53 Разработка веб-приложений на WordPress: Пер. с англ. / Б. Мессенленер, Д. Коулман. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2021. — 528 с.: ил.

ISBN 978-5-9775-6753-4

Подробно рассматривается создание веб-приложений на платформе WordPress, в том числе для мобильных устройств, принципы работы таких приложений. Описана структура каталогов и базы данных, приведены типы записей, метаданных и таксономий, перечислены основные классы и функции. Уделено внимание разработке собственной темы оформления с адаптивным дизайном. Рассказывается о типах пользователей и их ролях в архитектуре WordPress. Отдельная глава посвящена работе с API-интерфейсами, объектами и вспомогательными функциями, рассматриваются вопросы безопасности веб-приложений, принципы написания безопасного кода. Изучается REST API в WordPress, JavaScript-фреймворки, способы локализации приложений. Описаны принципы построения многосайтовых сетей, оптимизации и масштабирования. Рассматривается проект Gutenberg и его возможности.

Во втором издании авторы рассматривают новые функции и возможности актуальных версий WordPress. Все примеры кода из книги доступны на веб-сервисе GitHub.

Для веб-разработчиков

УДК 004.4'236
ББК 32.973.26-018

Группа подготовки издания:

Руководитель проекта	<i>Павел Шалин</i>
Зав редакцией	<i>Людмила Гауль</i>
Перевод с английского	<i>Михаила Райтмана</i>
Компьютерная верстка	<i>Натали Смирновой</i>
Оформление обложки	<i>Карины Соловьевой</i>

© 2021 BHV

Authorized Russian translation of the English edition of *Building Web Apps with WordPress 2nd edition* ISBN 9781491990087

© 2020 Brian Messenlehner and Jason Coleman

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Авторизованный перевод с английского языка на русский издания *Building Web Apps with WordPress 2nd edition* ISBN 9781491990087 © 2020 Brian Messenlehner and Jason Coleman

Перевод опубликован и продается с разрешения компании-правообладателя O'Reilly Media, Inc.

Подписано в печать 09 04 21

Формат 70×100^{1/16} Печать офсетная Усл печ л 42,57

Тираж 1000 экз Заказ № 8042

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20

Отпечатано в ПАО «Можайский полиграфический комбинат»

143200, Россия, г. Можайск, ул. Мира, 93

www.oaompk.ru тел. (495) 745-84-28, (49638) 20-685



ISBN 978-1-491-99008-7 (англ.)

ISBN 978-5-9775-6753-4 (рус.)

© Brian Messenlehner, Jason Coleman, 2020

© Перевод на русский язык, оформление
ООО "БХВ-Петербург", ООО "БХВ", 2021

Оглавление

Вступительное слово.....	15
Предисловие	17
Для кого предназначена эта книга	17
Для кого НЕ предназначена эта книга.....	18
Структура книги	18
О коде программ.....	20
Условные обозначения.....	20
Использование примеров программного кода.....	21
Благодарности	23
ГЛАВА 1. Создание веб-приложений с помощью WordPress	25
Что такое веб-сайт?	25
Что такое приложение?.....	25
Что такое веб-приложение?	25
Функции веб-приложения	26
Мобильные приложения.....	27
Прогрессивные веб-приложения	28
Зачем нужен WordPress?.....	29
Вы уже используете WordPress.....	29
С помощью WordPress легко управлять контентом.....	29
WordPress позволяет просто и безопасно управлять пользователями	30
Плагины	30
Гибкость важна	31
Частые обновления безопасности.....	31
Стоимость	32
Ответы на некоторые распространенные критические мнения о WordPress	32
Когда не следует использовать WordPress.....	35
Вы планируете лицензировать или продавать технологию своего сайта.....	35
Имеется другая платформа, которая приведет вас к цели быстрее.....	36
Гибкость не важна для вас	36
Ваше приложение должно работать в режиме реального времени.....	37
WordPress как фреймворк	37
WordPress и фреймворки Framework-View-Controller	38
Анатомия приложения WordPress.....	40
Что такое SchoolPress?.....	41
SchoolPress работает в многосайтовой сети WordPress.....	41
Бизнес-модель SchoolPress.....	41
Уровни участия и роли пользователей.....	42
Классы — это группы BuddyPress.....	42
Назначения — это CPT	42
Представления (подтип) CPT для назначений	43
Семестры являются таксономией для класса CPT	43

Департаменты являются таксономией для класса CPT	43
SchoolPress имеет один основной пользовательский плагин.....	43
В SchoolPress есть несколько других пользовательских плагинов	44
SchoolPress использует тему Memberlite.....	44
ГЛАВА 2. Основы WordPress.....	46
Структура каталогов WordPress	46
Корневой каталог	46
Структура базы данных WordPress	48
Таблица <i>wp_options</i>	49
Функции в каталоге /wp-includes/option.php.....	49
Таблица <i>wp_users</i>	51
Функции в каталоге /wp-includes/.....	52
Таблица <i>wp_usermeta</i>	55
Таблица <i>wp_posts</i>	59
Функции в каталоге /wp-includes/post.php	60
Таблица <i>wp_postmeta</i>	64
Функции из каталога /wp-includes/post.php	64
Таблица <i>wp_comments</i>	68
Функции в каталоге /wp-includes/comment.php	69
Таблица <i>wp_commentsmeta</i>	73
Функции из каталога /wp-includes/comment.php	73
Таблица <i>wp_terms</i>	75
Функции в каталоге /wp-includes/taxonomy.php.....	76
Таблица <i>wp_termmeta</i>	79
Таблица <i>wp_term_taxonomy</i>	81
Функции в каталоге /wp-includes/taxonomy.php.....	81
Таблица <i>wp_term_relationships</i>	82
Хуки: события и фильтры.....	83
События	84
Фильтры	85
Среды разработки и хостинг	86
Работа локально	86
Выбор веб-хостинга	87
Среды разработки, интеграции и доставки.....	87
Расширение WordPress	88
ГЛАВА 3. Использование плагинов WordPress	89
General Public License, версия 2	90
Установка плагинов WordPress	90
Создание собственного плагина.....	91
Структура файла плагина приложения.....	92
Каталог /adminpages/.....	93
Каталог /classes/.....	94
Каталог /css/.....	94
Каталог /js/	95
Каталог /images/	96
Каталог /includes/	96
Каталог /includes/lib/	97
Каталог /pages/.....	97

Каталоги /services/ и /scheduled/.....	98
Файл schoolpress.php	98
Дополнения к существующим плагинам.....	99
Случаи из практики и примеры.....	99
Цикл WordPress	100
Глобальные переменные WordPress	100
Бесплатные плагины	111
Admin Columns	111
Advanced Custom Fields	111
BadgeOS	112
Posts 2 Posts.....	112
Members	113
W3 Total Cache	113
Yoast SEO.....	113
Премиальные плагины	114
Gravity Forms	114
BackupBuddy.....	114
WP All Import.....	115
Плагины сообщества	115
BuddyPress	115
ГЛАВА 4. Темы.....	128
Темы и плагины	128
Где разместить код при разработке приложений.....	128
Где разместить код при разработке плагинов	129
Где разместить код при разработке тем.....	129
Иерархия шаблонов.....	130
Шаблоны страниц.....	131
Образец шаблона страницы	131
Использование хуков для копирования шаблонов	134
Когда следует использовать шаблон темы?	136
Функции WordPress для работы с темами.....	136
Использование переменной <i>locate_template</i> в плагинах.....	137
Файл style.css	139
Создание версий CSS-файлов вашей темы.....	140
Файл functions.php	141
Темы и CPT	142
Популярные фреймворки для разработки тем	142
Фреймворки тем WordPress.....	142
Сторонние фреймворки тем	144
Создание дочерней темы для Memberlite	144
Включение Bootstrap в тему вашего приложения	145
Меню.....	146
Навигационные меню	147
Динамические меню	148
Адаптивный дизайн.....	148
Определение устройства и дисплея с помощью CSS	149
Определение устройств и их свойств в JavaScript	150
Определение устройства в PHP	153
Последнее замечание по определению браузера	157

ГЛАВА 5. Пользовательские типы записей, метаданные записей и таксономия....	158
Типы сообщений по умолчанию и CPT.....	158
Страница.....	158
Публикация.....	158
Вложение.....	158
Редакции.....	159
Элемент меню навигации.....	159
Пользовательский CSS.....	159
Наборы изменений.....	159
Кеш oEmbed.....	159
Пользовательские запросы.....	160
Повторно используемые блоки.....	160
Определение и регистрация CPT.....	160
Функция <i>register_post_type(\$post_type, \$args)</i>	161
Что такое таксономия и как ее использовать?.....	168
Таксономии и метаданные постов.....	169
Создание пользовательских таксономий.....	171
Функция <i>register_taxonomy(\$taxonomy, \$object_type, \$args)</i>	171
Функция <i>register_taxonomy_for_object_type(\$taxonomy, \$object_type)</i>	174
Использование CPT и таксономий в ваших темах и плагинах.....	175
Тема архива и файлы шаблона Single.....	175
Старый добрый класс <i>WP_Query</i> и метод <i>get_posts()</i>	175
Метаданные и CPT.....	179
Функция <i>add_meta_box(\$id, \$title, \$callback, \$screen, \$context, \$priority, \$callback_args)</i>	179
Использование метаблоков в редакторе блоков Block Editor.....	182
Пользовательские классы-оболочки для CPT.....	183
Расширение класса <i>WP_Post</i> в сравнении с созданием класса-обертки.....	186
Зачем нужны классы <i>Wrapper</i> ?.....	186
Держите CPT и таксономии вместе.....	186
Держите все в классе-обертке.....	188
Классы <i>Wrapper</i> читаются лучше.....	190
ГЛАВА 6. Пользователи, их роли и возможности.....	191
Получение данных пользователей.....	192
Добавляем, обновляем и удаляем пользователей.....	194
Хуки и фильтры.....	198
Что такое роли и возможности?.....	199
Проверка роли и возможностей пользователя.....	200
Создание собственных ролей и возможностей.....	202
Расширение класса <i>WP_User</i>	204
Добавление полей регистрации и профиля.....	206
Настройка таблицы пользователей на административной панели.....	211
Плагины.....	213
Theme My Login.....	213
Hide the Admin Bar.....	213
Paid Memberships Pro.....	213
PMPro Register Helper.....	214
Members.....	214
WP User Fields.....	215

ГЛАВА 7. Работа с API-интерфейсами WordPress, объектами и вспомогательными функциями	216
API шорткодов	216
Атрибуты шорткода	217
Вложенные шорткоды	218
Удаление шорткодов	219
Другие полезные функции, связанные с шорткодами	220
API виджетов	220
Прежде чем добавить свой собственный виджет	221
Добавление виджетов	221
Определение области виджета	225
Встраивание виджета вне динамической боковой панели	227
Удаление виджетов с панели инструментов	229
Добавление собственного виджета на панель инструментов	231
API настроек	234
Вам действительно нужна страница настроек?	234
Не могли бы вы использовать вместо этого хук или фильтр?	235
Учет стандартов при добавлении настроек	236
Игнорирование стандартов при добавлении настроек	237
API перезаписи	238
Добавление правил перезаписи	238
Сброс правил перезаписи	239
Другие функции перезаписи	240
Функция <i>WP-Cron</i>	243
Добавление своих интервалов	245
Планирование единичных событий	246
Запуск заданий Cron с сервера	246
Использование только серверного Cron	247
Функция <i>WP Mail</i>	248
Отправка более приятных писем с помощью WordPress	249
API заголовка файла	250
Добавление заголовков файлов в ваши собственные файлы	253
Добавление новых заголовков в плагины и темы	254
Heartbeat API	255
ГЛАВА 8. Безопасность в WordPress	260
Почему это важно	260
Основные меры безопасности	261
Регулярно выполняйте обновление	261
Не используйте имя пользователя "admin"	261
Выбирайте надежный пароль	261
Примеры плохих паролей	262
Примеры хороших паролей	262
Усиление защиты в WordPress	263
Запретите администраторам редактировать плагины и темы	263
Измените префикс таблиц базы данных	263
Переместите в другое место файл <code>wp-config.php</code>	264
Не отображайте сообщения об ошибках авторизации	265
Не отображайте номер версии WordPress	265
Исключите возможность авторизации через страницу <code>wp-login.php</code>	266

Добавьте в файл *.htaccess кастомные правила, блокирующие доступ к каталогу wp-admin.....	267
SSL-сертификаты и HTTPS	268
Установка SSL-сертификата на сервере	268
Авторизация и доступ к панели администратора WordPress по протоколу SSL ...	271
Отладка проблем с протоколом HTTPS.....	272
"Атомный" способ устранения ошибок протокола SSL.....	273
Резервируйте все!	275
Сканируйте, сканируйте и еще раз сканируйте!.....	275
Полезные плагины для обеспечения безопасности.....	276
Плагины для блокировки спама.....	276
Плагины для резервного копирования.....	276
Плагины-брандмауэры/сканеры	277
Плагины для защиты авторизации и пароля	278
Написание безопасного кода	278
Проверяйте полномочия пользователей	278
Кастомные инструкции SQL	280
Валидация, санация и экранирование данных	280
Одноразовые коды	284
ГЛАВА 9. JavaScript-фреймворки и рабочий процесс	291
Что такое ECMAScript?.....	292
Что такое ES6?	293
Что такое ES9?	293
Что такое ESNext?	293
Что такое Ajax?	293
Что такое JSON?	293
jQuery и WordPress	294
Подключение других JavaScript-библиотек.....	295
Где следует размещать кастомный JavaScript-код.....	296
Ajax-вызовы в WordPress с использованием jQuery.....	297
Управление количеством Ajax-запросов.....	302
Heartbeat API	304
Инициализация.....	304
Клиентский JavaScript-код	305
Серверный PHP-код	306
Инициализация.....	307
Клиентский JavaScript-код	308
Серверный PHP-код.....	309
Ограничения WordPress в плане асинхронной обработки.....	310
JavaScript-фреймворки	311
Backbone.js.....	311
React	312
ГЛАВА 10. REST API в WordPress.....	314
Что такое REST API?.....	314
API	314
REST	315
JSON	315
HTTP	315

Зачем нужен REST API в WordPress?	318
Использование WordPress REST API версии 2	320
Обнаружение	320
Аутентификация	320
Маршруты и конечные точки	326
Запросы	326
Ответы	328
Добавление собственных маршрутов и конечных точек	329
Функция <i>register_rest_route(\$namespace, \$route, \$args, \$override)</i>	329
Настройка плагина WordPress Single Sign-On	330
Добавление маршрута <i>/wp-sso/v1/check</i>	330
Подключение к нашему плагину базовой аутентификации	332
Использование настроенной нами конечной точки для проверки учетных данных пользователя	333
Популярные плагины, использующие WordPress REST API	334
WooCommerce	335
BuddyPress	336
Paid Memberships Pro	338
ГЛАВА 11. Проект Gutenberg, блоки и кастомные типы блоков	342
Редактор системы WordPress	343
Плагин Classic Editor	344
Блоки для контента и дизайна	344
Блоки для представления функциональности	344
Создание собственных блоков	345
Пример простейшего блока	345
Использование кастомных блоков для разработки интерфейсов приложений	347
Активация редактора блоков для кастомных типов постов	347
Категории блоков	348
Блоки домашнего задания	349
Ограничение типа блоков до определенных кастомных типов постов	349
Ограничение кастомного типа постов до определенных блоков	350
Шаблоны блоков	351
Сохранение данных блока в метаданных поста	353
Советы	354
Активируйте режим отладки с помощью константы <i>WP_SCRIPT_DEBUG</i>	354
Задавайте версию скрипта с помощью функции <i>filemtime()</i>	355
Дополнительные советы	355
Глубже изучите JavaScript, Node.js и React	355
ГЛАВА 12. Многосайтовые сети в WordPress	357
Когда целесообразна многосайтовость?	357
Когда лучше отказаться от многосайтовости?	358
Альтернативы многосайтового режима	359
Множество авторов или категорий на одном и том же WordPress-сайте	359
Кастомные типы постов	359
Абсолютно самостоятельные сайты	360
Сервис обслуживания WordPress-сайтов	360
Мультиарендность	360
Настройка многосайтовой сети	360

Администрирование многосайтовой сети	363
Панель администратора	363
Сайты	363
Пользователи	364
Темы	365
Плагины	365
Настройки	365
Обновления	366
Структура базы данных многосайтовой сети	367
Общесетевые таблицы	367
Индивидуальные таблицы сайтов	369
Совместно используемые таблицы сайтов	370
Сопоставление доменов	370
Некоторые полезные плагины для многосайтового режима	371
Расширение User Registration для плагина Gravity Forms	372
Расширение Member Network Sites для плагина Paid Memberships Pro	372
Плагин More Privacy Options	372
Плагин Multisite Global Media	372
Плагин Multisite Plugin Manager	372
Плагин Multisite Global Search	373
Плагин Multisite Robots.txt Manager	373
Плагин NS Cloner: Site Copier	373
Плагин WP Multi Network	373
Основная функциональность многосайтовости	373
Переменная <i>\$blog_id</i>	374
Функция <i>is_multisite()</i>	374
Функция <i>get_current_blog_id()</i>	374
Функция <i>switch_to_blog(\$new_blog)</i>	375
Функция <i>restore_current_blog()</i>	375
Функция <i>get_blog_details(\$fields = null, \$get_all = true)</i>	376
Функция <i>update_blog_details(\$blog_id, \$details = array())</i>	377
Функция <i>get_blog_status(\$id, \$pref)</i>	378
Функция <i>update_blog_status(\$blog_id, \$pref, \$value)</i>	378
Функция <i>get_blog_option(\$id, \$option, \$default = false)</i>	378
Функция <i>update_blog_option(\$id, \$option, \$value)</i>	379
Функция <i>delete_blog_option(\$id, \$option)</i>	379
Функция <i>get_blog_post(\$blog_id, \$post_id)</i>	379
Функция <i>add_user_to_blog(\$blog_id, \$user_id, \$role)</i>	380
Функция <i>wpmu_delete_user(\$user_id)</i>	380
Функция <i>create_empty_blog (\$domain, \$path, \$weblog_title, \$site_id = 1)</i>	381
Не упомянутые здесь функции	381
ГЛАВА 13. Локализация приложений WordPress	382
Нужна ли локализация вашему приложению?	382
Как выполняется локализация в WordPress	383
Определение локали в WordPress	383
Текстовые домены	384
Настройка текстового домена	384
Подготовка строк с помощью функций перевода	386
Функция <i>__(\$text, \$domain = "default")</i>	387
Функция <i>_e(\$text, \$domain = "default")</i>	387

Функция <code>_x(\$text, \$context, \$domain = "default")</code>	388
Функция <code>_ex(\$title, \$context, \$domain = "default")</code>	388
Сочетание перевода с экранированием.....	389
Создание и загрузка файлов перевода.....	389
Организация файлов локализации.....	390
Генерирование файла *.pot	390
Создание файла *.po	392
Создание файла *.mo	392
GlotPress	392
Использование GlotPress для ваших плагинов и тем в репозитории	
WordPress.org.....	393
Создание собственного сервера GlotPress	393
ГЛАВА 14. Оптимизация и масштабирование WordPress	394
Терминология	394
Источник или внешнее окружение?	396
Тестирование	396
Что следует тестировать.....	397
Панель отладки браузера Chrome	399
Инструмент Site Health системы WordPress	401
Apache Bench	402
Siege.....	409
W3 Total Cache.....	410
Настройки страничного кэширования	411
Минимизация	413
Кэширование базы данных.....	414
Объектное кэширование.....	414
Сети доставки контента.....	415
GZIP-сжатие	415
Хостинг.....	415
Хостинги, специально предназначенные для WordPress-сайтов.....	416
Развертывание собственного сервера	416
Выборочное кэширование	430
API для работы с транзиентами	431
Транзиенты в многосайтовом режиме	434
Повышение производительности с помощью JavaScript-кода	434
Кастомные таблицы.....	436
Действие в обход WordPress.....	438
ГЛАВА 15. Электронная коммерция.....	440
Выбор плагина	440
WooCommerce	440
Paid Memberships Pro	443
Easy Digital Downloads.....	444
Платежные системы	447
Торговые счета.....	448
Настройка модели SaaS с помощью Paid Memberships Pro	449
Модель SaaS	449
ГЛАВА 16. Мобильные приложения на платформе WordPress.....	464
Сценарии использования мобильных приложений.....	464

Нативные и гибридные мобильные приложения.....	465
Что такое нативное мобильное приложение?.....	465
Что такое гибридное мобильное приложение?	466
Почему стоит создавать гибридные приложения вместо нативных?.....	466
Cordova.....	467
Ionic Framework.....	471
Приложение-обертка	473
AppPresser	473
ГЛАВА 17. RHP-библиотеки, интеграция веб-сервисов и миграция	
с других платформ.....	488
RHP-библиотеки	488
Генерация и модификация изображений	489
Генерация PDF	491
Геолокация и геотаргетинг.....	494
Сжатие и архивация данных	496
Инструменты для разработки	500
Внешние API-интерфейсы и веб-сервисы	502
Elasticsearch	502
ElasticPress by 10up	502
Google Vision	503
Google Maps.....	503
Google Translate.....	504
Twilio.....	504
Другие популярные API-интерфейсы	505
Миграция.....	506
Миграция между серверами.....	507
Миграция между платформами	508
Руководство по привязке данных	510
ГЛАВА 18. Взгляд в будущее.....	511
Оглядываясь назад.....	511
REST API.....	512
Плагины WordPress будут уделять больше внимания API-интерфейсам.....	512
"Обезглавленные" версии WordPress	512
GraphQL	513
Gutenberg	514
Интерфейс администратора перейдет на React/Gutenberg	514
Gutenberg будет применяться для редактирования контента	
на клиентской стороне WordPress	514
Шаблоны блоков заменят темы оформления	514
Блоки заменят плагины	515
Доля WordPress на рынке будет колебаться	515
WordPress станет более популярной платформой для мобильной разработки.....	516
WordPress будет оставаться хорошим выбором для разработки любого рода	
приложений.....	516
Об авторах.....	517
Предметный указатель.....	518

Вступительное слово

В течение последних нескольких лет я проводил от 15 до 24 выходных каждый год, встречаясь с разработчиками программного обеспечения по всей стране. В итоге после примерно 25 тыс. диалогов я убедился в трех простых фактах:

- ◆ Веб-приложения встречаются повсеместно.

Трудно представить жизнь без веб-приложений. Сегодня мы управляем нашими банковскими счетами онлайн. Мы бронируем отели и отдых онлайн с помощью веб-приложений; мы платим наши налоги онлайн в чем-то вроде приложения; мы записываемся в парикмахерскую перед визитом, используя то, что кажется простой формой, но это тоже приложение. Веб-приложения вездесущи.

- ◆ Веб-разработчики чувствуют себя совершенно неспособными создавать веб-приложения.

Если вы поговорите с обычными пользователями, то каждый из них скажет, что у него есть друг-разработчик — гений, который может делать что угодно с компьютером. Но если вы потратите время на разговоры непосредственно с разработчиками, то обнаружите, что они видят мир совсем по-другому.

Они смотрят на веб-приложения, которые считают "реальными", и на компании, стоящие за ними, и убеждены, что люди, которые работают "там", знают о создании веб-приложений то, что никому не известно.

Разработчики убеждены, что в одиночку могут создавать только небольшие приложения, и они не знают, что же им нужно для создания такого приложения, которое станет невероятно популярным.

- ◆ Чего не хватает, так это немного информации и гораздо больше уверенности.

Когда вы разговариваете с разработчиками о веб-приложениях, то быстро обнаруживаете, что отсутствующий элемент — это уверенность, ее не хватает больше всего. Написать программный код не сложно, если вы знаете, что необходимо делать. Но это не значит, что разработчикам не нужно понимать процесс глубже.

Наши беседы проходили по всей стране на местных конференциях под названием WordCamps. Это общественные мероприятия, которые обычно посещают от 200 до 500 участников — в равной степени как разработчики, так и пользователи приложений. Но все они используют (или намереваются это сделать) продукт веб-публикации под названием WordPress.

На WordPress, если вы никогда не слышали о нем, работает треть интернет-сайтов на планете. Он в основном служит в качестве платформы для публикации контента. Но на самом деле с помощью WordPress можно также запустить ваше веб-приложение.

Об этом пойдет речь на страницах данной книги.

Мне нравится в этой книге, что первые главы достаточно подробны. Это помогает читателям медленно погрузиться в предмет и придает уверенность. И я знаю, насколько важным это обстоятельство будет для вас. Образно выражаясь, книга создает "дорожную карту", показывая, как добраться от того места, где вы находитесь, до того места, где вы хотите оказаться.

Но ценность этой книги заключается еще и в том, что она глубже "вдавливает" вас в детали каждой строки кода, которую вам нужно написать. Что-то будет новым для вас, что-то — нет. Но чего вы здесь точно не найдете, так это абстрактных описаний, которые никак не объясняются и оставляют открытым вопрос, как применить полученные знания.

Брайан и Джейсон знакомят вас с каждой составной частью WordPress, которую вы должны знать, чтобы применить его в качестве основы веб-приложения. Сегодня существуют курсы программирования, ориентированные на преподавание именно этого материала. К счастью, вам не придется платить за подобные курсы, которые стоят намного дороже, чем эта книга. И это, пожалуй, лучшая реклама данной книги.

До сих пор вы, возможно, думали, что есть два типа разработчиков программ. Первые — те, кто создает сложные аппаратно-ориентированные приложения, которые вам не под силу. Вторые, как и вы, разрабатывают небольшие простые приложения.

Когда вы прочитаете эту книгу до конца, изучите теорию и "поиграете" с программным кодом, когда вы заставите его работать, то поймете, что существует только один вид разработчика: тот, который способен создавать все, что только можно представить, и вы станете именно таким разработчиком.

Тот факт, что вы сейчас читаете эти строки, уже подтверждает сказанное. Теперь вам пора идти дальше и реализовать свой потенциал.

Крис Лема, вице-президент по продуктам и инновациям в Liquid Web

Предисловие

На момент написания этой книги WordPress поддерживает 32% всех сайтов в Интернете, и это число растёт. Многие разработчики хотят расширить возможности своих сайтов WordPress, но считают, что им нужно перейти к более традиционной среде приложений, такой как Ruby on Rails, Symfony, Yii или Laravel, для создания "настоящих" веб-приложений. Такое мнение неверно, и мы здесь, чтобы исправить это заблуждение.

Несмотря на то, что WordPress изначально был программным обеспечением для ведения блогов и в настоящее время существует в основном как система управления контентом, постепенно он превратился в гибкую и функциональную платформу для создания веб-приложений. Эта книга покажет вам, как использовать WordPress в качестве фреймворка для создания *любого* веб-приложения, большого или маленького.

Для кого предназначена эта книга

Эта книга будет наиболее полезна для разработчиков WordPress, желающих создавать более "тяжелые" приложения, и для программистов PHP, имеющих некоторый опыт работы с WordPress, ищущих среду создания приложений на основе PHP.

Коммерческие разработчики плагинов и те, кто работает над крупными распределёнными проектами WordPress, также найдут полезными концепции и методы этой книги.

Если вы PHP-программист или независимый от языка разработчик, использующий другую платформу, и пользуетесь огромной библиотекой плагинов и тем WordPress, то вы возможно удивитесь, узнав, насколько хорошо WordPress может работать как фреймворк. Чтение этой книги и выполнение упражнений из нее могут изменить вашу трудовую жизнь в лучшую сторону.

Мы предполагаем, что читатели имеют общие сведения о программировании на PHP. Вы также должны обладать базовыми знаниями HTML и CSS, а также быть знакомыми с MySQL и SQL-запросами. Понимание основ JavaScript и программирования jQuery понадобится в *главе 9* и при рассмотрении связанных примеров.

Для кого НЕ предназначена эта книга

Эта книга не для тех, кто хочет научиться работать с WordPress в качестве пользователя. В ней изложены краткие введения о стандартной функциональности WordPress, но мы предполагаем, что вы уже знакомы с WordPress с точки зрения пользователя.

Эта книга не предназначена для непрограммистов. Несмотря на то, что можно создавать очень функциональные веб-приложения, просто комбинируя и конфигурируя множество плагинов, доступных для WordPress, эта книга написана для разработчиков, создающих свои собственные плагины и темы для новых веб-приложений.

Эта книга не научит вас программировать, а скорее научит программировать "путем WordPress".

Структура книги

Мы надеемся, что благодаря этой книге вы изучите методы программирования и проектирования, а также освоите лучшие практики для разработки сложных приложений с использованием WordPress.

Глава 1 определяет, что мы подразумеваем под "веб-приложением", а также описывает, почему следует (или не следует) создавать веб-приложения на WordPress, и как сравнивать WordPress с другими фреймворками. Мы также представляем SchoolPress, приложение WordPress, которое служит нам в качестве примера на протяжении всей книги.

Глава 2 охватывает основы WordPress. Мы просматриваем различные каталоги базовой установки WordPress. Мы также объясняем каждую таблицу базы данных, созданную WordPress, какие данные хранятся в каждой и какие функции WordPress отображаются на эти таблицы. Даже если вы опытный разработчик WordPress, вы можете кое-что узнать из этой главы, и мы рекомендуем вам прочитать ее.

Глава 3 повествует все о плагинах. Что это такое? Как создавать свои собственные плагины? Как вы должны структурировать основной плагин вашего приложения? Когда вы должны использовать сторонние плагины или свои собственные?

Глава 4 — это все о темах. Как работают темы? Как темы отображаются на представления в типичной структуре "модель – представление – контроллер" (MVC)? Какой код должен входить в вашу тему, а какой — в плагины? Еще мы рассмотрим использование фреймворков тем и UI, а также познакомимся с основами адаптивного дизайна.

Глава 5 описывает пользовательские типы записей и таксономию. Мы рассмотрим стандартные типы записей, встроенные в WordPress, выясним, почему вам может потребоваться создать свои собственные, а затем опишем, как это сделать. Мы также расскажем о метаданных и таксономиях публикаций, о том, для чего каждый из них подходит, и о том, как создавать собственные таксономии и сопоставлять их с

вашими типами записей. Наконец, мы покажем, как создавать классы-обертки для ваших типов записей, чтобы оптимизировать ваш код с помощью объектно-ориентированного программирования (ООП).

Глава 6 посвящена пользователям, их роли и возможностям. Мы покажем, как добавлять, обновлять и удалять пользователей программно и как работать с метаролями, ролями и правами пользователей. Мы также покажем, как расширить класс `WP_User` для ваших пользовательских архетипов, таких как "клиенты" и "учителя", чтобы лучше организовать ваш код на основе методов ООП.

Глава 7 охватывает некоторые из более полезных API-интерфейсов WordPress и вспомогательных функций, которые не вписываются в остальную часть книги, но все еще важны для разработчиков, создающих веб-приложения с помощью WordPress.

Глава 8 содержит все сведения о защите ваших приложений WordPress, плагинов и тем.

Глава 9 охватывает использование JavaScript и Ajax в вашем приложении WordPress. Мы расскажем о правильном способе включения JavaScript в WordPress и о том, как создать асинхронное поведение в вашем приложении.

Глава 10 рассказывает о REST API для WordPress и о том, как применить его для интеграции WordPress с внешними приложениями.

Глава 11 посвящена описанию редактора блоков и созданию своих собственных блоков.

Глава 12 рассматривает многосайтовые сети WordPress, включая их настройку и обстоятельства, которые следует учитывать при разработке для сети.

Глава 13 описывает локализацию ваших плагинов и тем для WordPress, в том числе вопрос, как подготовить код для перевода и как создавать и использовать файлы перевода.

Глава 14 раскрывает, как оптимизировать и масштабировать WordPress для больших веб-приложений. Мы рассмотрим, как проверить производительность вашего приложения WordPress и укажем наиболее популярные методы ускорения и масштабирования сайтов под управлением WordPress.

Глава 15 посвящена созданию приложения для интернет-торговли. Мы рассмотрим различные типы доступных плагинов для электронной коммерции и способы выбора между ними. Затем мы подробно опишем, как с помощью WordPress можно обрабатывать платежи и управлять учетными записями веб-приложения SaaS (software as a service).

Глава 16 раскрывает, как использовать WordPress для запуска собственных приложений на мобильных устройствах, создавая оболочки приложений для iOS и Android.

Глава 17 описывает некоторые сторонние библиотеки PHP, службы и API, которые часто встречаются в веб-приложениях, а также способы их интеграции с WordPress, включая полную миграцию.

Глава 18 предсказывает будущее WordPress, описывает, какие приложения мы ожидаем увидеть на WordPress, какие обновления мы ожидаем для WordPress, а также на какие инструменты и платформы следует обратить внимание в будущем.

О коде программ

Все примеры из этой книги вы можете найти по адресу github.com/bwawwp. Обратите внимание, что эти примеры кода были написаны для того, чтобы наиболее четко передать концепции, которые мы рассмотрим в книге. Чтобы улучшить удобочитаемость листингов программ, мы часто игнорируем лучшие практики стандартов программирования (oreil.ly/xw3dV), безопасность и локализацию (которые мы рассмотрим в главах 8 и 13) или не рассматриваем некоторые особые случаи. Вы должны иметь это в виду, прежде чем использовать какие-либо примеры в своем рабочем коде.

Образец приложения SchoolPress доступен по адресу schoolpress.me, с открытым исходным кодом для этого сайта по адресу oreil.ly/6Lbax.

Условные обозначения

В этой книге приняты следующие обозначения:

Курсивный шрифт — указывает на новые термины, имена файлов и расширения файлов.

Полужирный шрифт — им оформлены URL-адреса и адреса электронной почты.

Моноширинный шрифт — используется для листингов программ, а также в абзацах для ссылки на элементы программы, такие как имена переменных или функций, базы данных, типы данных, переменные среды, операторы и ключевые слова.

Полужирный моноширинный шрифт — указывает на команды или другой текст, который должен быть набран пользователем.

Курсивный моноширинный шрифт — показывает текст, который следует заменить предоставленными пользователем значениями или значениями, определенными контекстом.



Данный элемент обозначает подсказку или совет.



Данный элемент обозначает примечание или замечание.



Данный элемент обозначает предупреждение или предостережение.

Использование примеров программного кода

Как было указано в разделе "О коде программ", все примеры программного кода в этой книге можно найти по адресу github.com/bwawwp.

Если у вас появится технический вопрос или проблема с использованием примеров кода, отправьте электронное письмо по адресу bookquestions@oreilly.com.

Эта книга призвана помочь вам выполнить свою работу. В общем случае, если пример кода приведен в книге, то вы можете вставить его в свои программы и документацию. Вам не нужно обращаться к нам за разрешением, если вы не воспроизводите значительную часть листинга программы. Например, написание программы, которая содержит несколько фрагментов кода из этой книги, не требует разрешения. Для продажи или распространения примеров из книг O'Reilly потребуется разрешение. Чтобы ответить на какой-либо вопрос, сославшись на эту книгу и приведя пример кода, разрешения не нужно. Включение значительного объема кода программ из этой книги в документацию вашего продукта требует разрешения.

Для нас важно, если при ссылке вы укажете атрибуты издания, но мы этого не требуем. Атрибуты обычно включают название книги, автора, издателя и ISBN. Например: "*Создание веб-приложений с WordPress*, второе издание, Брайан Мессенленер и Джейсон Коулман (O'Reilly). Все права принадлежат Брайану Мессенленеру и Джейсону Коулману, 978-1-491-99008-7 2020."

Если вы считаете, что использование примеров кода требует разрешения, свяжитесь с нами по адресу permissions@oreilly.com.

Благодарности

От Брайана: спасибо Джейсону Коулману и Мэтту Мулленвегу, я не смог бы написать эту книгу без их помощи! Особая благодарность Алисии Янг за то, что она остается в курсе событий в O'Reilly Media. Выражаю также благодарность нашим техническим экспертам за то, что все в книге является верным. Спасибо Скотту Болинжеру из AppPresser.com и Джеффу Уорлею из AlphaWeb.com за то, что терпели меня. Привет семье и друзьям, которые всегда были рядом со мной и никогда не переставали верить в меня. Больше всего я благодарен своим детям: Дали, Брайану-младшему, Нине, Кэму и Акселю Мессенленерам — они дают мне цель, и без них я, вероятно, даже не знал бы, что такое WordPress.

От Джейсона: спасибо моему соавтору Брайану за то, что он попросил меня написать эту книгу вместе с ним. Спасибо нашим первым редакторам Меган и Аллисон за то, что они следили за нами и помогали нам оставаться верными нашему первоначальному видению. Спасибо Алисии Янг за редактирование второго издания этой книги и за всю проделанную работу с нашими объяснениями "WP Drama". Выражаю благодарность всем замечательным техническим редакторам, которые были у нас в обеих редакциях книги: Сэм Хотчкисс, Питер Макинтайр, Пиппин Уильямсон, Джон Джеймс Джейкоби и Эндрю Лима. Спасибо Фредерику Таунсу за его отзывы и вклад в нашу главу по оптимизации и масштабированию. Спасибо Крису Леме за замечательный перевод этой книги, его отзывы о книге и советы в целом. Благодарю всех в сообществе WordPress, кто ответил на все мои случайные твиты и, возможно, знал или не знал, что они помогают мне написать эту книгу. Спасибо моей жене, Ким, за то, что она поддержала меня, как всегда, во время еще одного приключения в нашей жизни. Спасибо моей дочери Марин за то, что она скучала по мне, когда я отсутствовал, чтобы писать, и моему сыну, Исааку, за то, что постоянно спрашивал меня, "закончил ли я уже книгу". И последнее, но не менее важное — спасибо моей семье, которая всегда поддерживала меня во время написания книги: мама, папа, Джереми и Нана Мэн с нетерпением ждут, когда станут первыми непрограммистами, прочитавшими эту книгу.

Создание веб-приложений с помощью WordPress

Эта книга поможет вам создать что угодно с помощью WordPress: веб-сайты, темы, плагины, веб-сервисы и веб-приложения. Мы решили сосредоточиться на веб-приложениях, потому что вы можете рассматривать их как супер-сайты, использующие все методы, которые мы опишем.

Многие считают, что WordPress недостаточно мощен или не предназначен для создания веб-приложений; далее мы еще вернемся к этому вопросу. Мы много лет создавали веб-приложения на WordPress и знаем, что с его помощью вы можете создавать масштабируемые приложения.

В этой главе мы начнем с определения того, что такое веб-приложение, а затем выясним, почему WordPress является отличным фреймворком для их создания. Мы также опишем некоторые ситуации, в которых применение WordPress *не будет* лучшим способом для создания вашего веб-приложения.

Что такое веб-сайт?

Вы знаете, что такое веб-сайт: набор из одной или нескольких веб-страниц, содержащих информацию, доступ к которой осуществляется через веб-браузер.

Что такое приложение?

Нам нравится определение в Википедии (oreil.ly/2DUK1): "Прикладное программное обеспечение (сокращенно приложение) — это программное обеспечение, предназначенное для выполнения группы скоординированных функций, задач или действий на благо пользователя".

Что такое веб-приложение?

Веб-приложение — это всего лишь приложение, запускаемое через веб-браузер.

Обратите внимание, что в некоторых веб-приложениях технология браузера скрыта, например, когда вы интегрируете веб-приложение в собственное приложение для Android или iOS, запускаете веб-сайт как приложение в Google Chrome или активизируете приложение с помощью Adobe AIR. Однако внутри этих приложений по-прежнему существует система парсинга HTML, CSS и JavaScript.

Вы также можете думать о веб-приложении как о *веб-сайте с некоторым дополнительным функционалом приложения*. Не существует точной разделительной линии, за которой веб-сайт становится веб-приложением. Это один из тех случаев, когда данное обстоятельство совершенно очевидно для вас.

Мы *можем* лишь объяснить некоторые функции веб-приложения, дать вам несколько примеров, а затем попытаться придумать сокращенное определение, чтобы вы в целом понимали, о чем мы говорим, когда употребляем этот термин в книге.



При чтении этой книги вам встретятся ссылки на SchoolPress. SchoolPress — это веб-приложение, которое мы создаем, чтобы помочь школам и преподавателям управлять своими учениками и учебными программами. Все примеры кода направлены на функциональность, которая может существовать в SchoolPress. Более подробно об общей концепции SchoolPress мы поговорим далее в этой главе.

Функции веб-приложения

Далее перечислены некоторые функции, обычно связанные с веб-приложениями и приложениями в целом. Чем больше этих функций есть на веб-сайте, тем целесообразнее считать его веб-приложением¹.

◆ Интерактивные элементы

Типичный веб-сайт включает в себя навигацию по страницам, прокрутку и нажатие гиперссылок. Веб-приложения также могут иметь ссылки и прокрутку, но чаще встречаются другие методы навигации по приложению.

Веб-сайты с формами предлагают функционал совершения транзакций. Примером может служить форма обратной связи на веб-сайте или форма заявки на странице вакансий на веб-сайте компании. Формы позволяют пользователям взаимодействовать с сайтом, используя нечто большее, чем клик.

Веб-приложения будут иметь еще больше элементов интерактивного пользовательского интерфейса (UI — user interface). Примеры включают в себя панели инструментов, элементы перетаскивания, редакторы форматированного текста и ползунки.

◆ Задачи, а не содержание

Помните, что веб-приложения "разработаны, чтобы помочь пользователю выполнять определенные задачи". Пользователи Google Maps получают маршруты проезда. Пользователи Gmail пишут электронные письма. Пользователи Trello управляют списками. Пользователи SchoolPress добавляют комментарии по теме урока.

Некоторые приложения по-прежнему ориентированы на контент. Типичный сеанс Facebook или Twitter почти на 90% состоит из чтения. Тем не менее сами

¹ На многие идеи в этом разделе повлияли сообщения в блоге: "Что такое веб-приложение?" Доминика Хазаэль-Массо (bit.ly/wiawa) и Боба Бэксли (bit.ly/wiawa2).

приложения представляют способ просмотра контента, отличный от обычного просмотра веб-страниц.

◆ *Логины*

Вход в систему и учетные записи позволяют веб-приложению сохранять информацию о своих пользователях. Эта информация предназначена для облегчения основных задач приложения и обеспечения постоянного взаимодействия. При входе в систему пользователи SchoolPress могут видеть, какие сообщения являются неп прочитанными. У них также есть имя пользователя, которое идентифицирует их деятельность в приложении.

Веб-приложения также могут иметь уровни пользователей. В SchoolPress будут администраторы, контролирующие внутреннюю работу приложения, учителя, управляющие процессом обучения и ученики, участвующие в дискуссиях в классе.

◆ *Возможности устройства*

Веб-приложения, работающие на вашем телефоне, могут получить доступ к вашей камере, адресной книге, внутреннему хранилищу и информации о местоположении GPS. Веб-приложения, функционирующие на персональном компьютере, могут иметь доступ к веб-камере или локальному жесткому диску. Одно и то же веб-приложение может реагировать по-разному в зависимости от устройства, на котором оно запущено. Веб-приложения будут настраиваться для разных размеров экранов, их разрешений и возможностей.

◆ *Работа в автономном режиме*

Всегда, когда это возможно, желательно, чтобы ваши веб-приложения работали в автономном режиме. Конечно, интерактивность Интернета главным образом определяет "веб" часть веб-приложения, но сайт, который все еще работает, когда вы проезжаете через туннель, будет больше походить на приложение.

С Gmail вы можете создавать черновики писем в автономном режиме. Evernote позволяет создавать заметки офлайн, а затем синхронизировать их с Интернетом после восстановления подключения.

◆ *Мэшапы*

Веб-приложения могут связывать одно или несколько веб-приложений вместе. Веб-приложение может использовать различные веб-сервисы и API для передачи и извлечения данных. У вас может быть веб-приложение, которое получает информацию о местоположении, такую как долгота и широта, из Twitter и отправлять и отмечать его на карте Google Map.

Мобильные приложения

С тех пор, как первое издание этой книги было опубликовано еще в 2012 году, веб-приложения, в частности мобильные приложения, получили широкое распространение. На большинстве веб-сайтов мобильные устройства уже преобладают

над ПК и являются крупнейшим источником трафика (Источник: Perficient, Inc. (oreil.ly/N92kX)).

В 2012 году типичное веб-приложение выглядело как Basecamp — менеджер проектов, доступ к которому осуществлялся через веб-браузер на вашем компьютере. В 2019 году типичное веб-приложение выглядит как Twitter — приложение для коммуникации, доступ к которому можно получить с помощью телефона iOS или Android.

Поскольку в большинстве случаев многие из пользователей будут получать доступ к вашим веб-сайтам и приложениям на мобильном устройстве, при разработке веб-приложений мы придерживаемся взгляда "прежде всего для мобильных устройств". О том, как заставить ваши приложения WordPress работать на мобильных устройствах, речь пойдет в *главе 16*. А об основах адаптивного дизайна и о том, как заставить ваши веб-сайты правильно отображаться на экранах любого размера мы расскажем в *главе 4*.

Прогрессивные веб-приложения

Прогрессивные веб-приложения (PWA) — это веб-сайты, реализующие преимущества современных функций браузера, которые ведут себя как собственные приложения для Android, iOS или ПК. В частности, веб-сайты, которые используют *сервис-воркеры* (service workers) для работы в автономном режиме, содержат файл манифеста веб-приложения для определения приложения в операционной системе (ОС) и отвечают нескольким другим требованиям, поэтому могут быть запущены как приложения прямо из браузера.

Идея PWA была разработана командой Google Chrome, но теперь поддерживается на iOS и в большинстве современных веб-браузеров. Плагин для поддержки PWA (oreil.ly/gThAQ) разрабатывается для работы основных функций PWA в WordPress core. С помощью этого плагина вы можете превратить ваш сайт WordPress в PWA, и это хорошая идея. Но на самом деле создание PWA — это скорее образ мышления, а не простой переход. Аналогично "функциям веб-приложения", которые мы только что описали, главный сайт PWA Google (oreil.ly/FawTK) содержит контрольный список функций, необходимых в большинстве PWA (oreil.ly/HBuTg). Укажем следующие базовые функции:

- ◆ Сайт обслуживается по HTTPS.
- ◆ Страницы адаптированы для планшетов и мобильных устройств.
- ◆ Все URL-адреса приложения загружаются в автономном режиме.
- ◆ Метаданные предоставляются для добавления на главный экран.
- ◆ Первая загрузка быстрая даже с 3G.
- ◆ Сайт работает независимо от браузера.
- ◆ Переход между страницами не ощущается как работа с сетью.
- ◆ У каждой страницы есть URL.

В дополнение к базовым функциям есть контрольный список элементов для "примерных" PWA, который охватывает пользовательский опыт (UX) и производительность. Инструмент Google Lighthouse (oreil.ly/GwEkb) предоставляет автоматизированные тесты и отчеты для соответствия критериям PWA. Даже разработчики полностью нативных приложений или приложений для браузера, могут воспользоваться некоторыми советами из контрольных списков PWA и отчетов Lighthouse.

Зачем нужен WordPress?

Ни один язык программирования или программный инструмент не подойдет для любой разработки. Мы еще коснемся вопроса, почему вы *не* захотите использовать WordPress, но сейчас давайте рассмотрим некоторые ситуации, в которых создавать веб-приложения целесообразно именно с помощью WordPress.

Вы уже используете WordPress

Если вы уже используете WordPress для своего основного сайта, то можете просто добавить плагин, который вам необходим. В WordPress есть отличные плагины для электронной торговли (WooCommerce), форумов (bbPress), сайтов с подпиской (Paid Memberships Pro), функций социальных сетей (BuddyPress) и геймификации (BadgeOS).

Встраивание приложения в существующий сайт WordPress экономит ваше время и упростит работу для ваших пользователей. Итак, если ваше приложение достаточно простое, вы можете создать собственный плагин на своем сайте WordPress для программирования функциональности вашего веб-приложения.

Когда вы довольны своим сайтом на WordPress, не поддавайтесь искушению, если люди говорят, что вам нужно перейти на что-то другое, чтобы добавить определенные функции на ваш сайт. Это, скорее всего, неправда. Вам не нужно выбрасывать всю работу, которую вы уже проделали в WordPress, и последующие доводы — это веские причины придерживаться WordPress.

С помощью WordPress легко управлять контентом

Разработанный сначала как платформа для ведения блогов, с введением пользовательских типов записей (англ. CPT — Custom Post Type) в версии 3.0 WordPress развился в полностью функциональную систему управления контентом (англ. CMS — Content Management System). Любая страница или сообщение может быть отредактировано администратором через панель инструментов, доступ к которой можно получить через ваш веб-браузер. О работе с CPT вы узнаете в *главе 5*.

WordPress упрощает добавление и редактирование контента с помощью редактора WYSIWYG (What You See Is What You Get — "Что видишь, то и получаешь"), поэтому вам не нужно привлекать веб-дизайнеров каждый раз, когда вы хотите внести простые изменения в свой сайт. Вы также можете создавать собственные меню и элементы навигации для своего сайта, не касаясь программного кода.

Если ваше веб-приложение сфокусировано на фрагментах контента (например, наше приложение SchoolPress ориентировано на назначение пользователям заданий и их обсуждение), API пользовательских типов постов для WordPress (описанный в главе 5) позволяет легко настроить этот пользовательский контент и управлять им.

Даже приложения, более ориентированные на задачи, обычно имеют несколько страниц с информацией, документацией и продажами. Использование WordPress для вашего приложения даст вам возможность управления приложением и всем вашим контентом из одного места.

WordPress позволяет просто и безопасно управлять пользователями

В WordPress есть все необходимое для добавления на сайт как администраторов, так и конечных пользователей.

Помимо управления доступом к контенту, система ролей и возможностей в WordPress расширяема и позволяет вам контролировать, какие *действия* доступны для определенных групп пользователей. Например, по умолчанию пользователи с ролью участника могут *добавлять* новые сообщения, но не могут *публиковать* их. Точно так же вы можете создавать новые роли и возможности для управления доступом к вашим пользовательским функциям.

Вы можете использовать плагины, такие как Paid Memberships Pro, чтобы расширить встроенное управление пользователями и назначать членов разных уровней и контролировать доступ пользователей к контенту. Например, вы можете создать уровень, чтобы предоставить участникам с платным аккаунтом доступ к премиум-контенту на вашем сайте WordPress.

Плагины

В репозитории WordPress (wordpress.org/plugins/) имеется более 55 тыс. бесплатных плагинов. Существует множество дополнительных плагинов, как бесплатных, так и коммерческих, на различных сайтах в Интернете. Если у вас появляется идея для расширения вашего сайта, велика вероятность, что для этого уже есть плагин, который сэкономит ваше время и деньги.

Существует несколько необходимых плагинов, которые мы задействуем практически на каждом создаваемом нами сайте и веб-приложении.

Для большинства разрабатываемых веб-сайтов вы, наверное, хотите кэшировать вывод для ускорения просмотра, использовать такие инструменты, как Google Analytics для отслеживания посетителей, создавать карты сайта и настраивать набор страниц для поисковой оптимизации (англ. SEO — Search Engine Optimization), а также выполнять ряд других общих задач.

Есть много хорошо поддерживаемых плагинов для всех этих функций. Мы предлагаем наши любимые в этой книге. Вы можете найти их список на веб-сайте этой книги (bwwwp.com/plugins/).

Гибкость важна

WordPress — полноценный фреймворк, способный на многое. Кроме того, WordPress построен на основе технологий PHP, JavaScript и MySQL, поэтому все, что вы можете встроить в PHP/MySQL (а это почти все), может быть достаточно легко встроено в ваше приложение WordPress.

WordPress и PHP/MySQL в целом не идеально подходят для любой задачи, но они пригодны для широкого круга задач. Наличие одной платформы, которая будет расти вместе с вашим бизнесом, позволит вам быстрее выполнять задачи и пере-страиваться. Например, вот типичный пример сайта запуска стартапа Lean, работающего на WordPress:

- ◆ Объявите о своем стартапе с помощью одностраничного сайта.
- ◆ Добавьте форму для сбора адресов электронной почты.
- ◆ Добавить блог.
- ◆ Сосредоточьтесь на SEO и оптимизируйте весь контент.
- ◆ Отправляйте посты блога в Twitter и Facebook.
- ◆ Добавьте форумы.
- ◆ Используйте плагин Paid Memberships Pro, чтобы позволить участникам платить за доступ.
- ◆ Добавьте пользовательские формы, инструменты и поведение приложения для участников с платной подпиской.
- ◆ Обновите пользовательский интерфейс, используя JavaScript и фреймворки.
- ◆ Настройте сайт и сервер для масштабирования.
- ◆ Локализируйте сайт/приложение для разных стран и языков.
- ◆ Добавьте поддержку Progressive Web App.
- ◆ Запустите обертки для iOS и Android для приложения.

Суть, почему нужно идти по этому пути, состоит в том, что на каждом этапе у вас есть *одна и та же база данных пользователей и одна и та же платформа разработки*.

Частые обновления безопасности

Тот факт, что WordPress используется на миллионах сайтов, делает его целью хакеров, пытающихся найти уязвимости в его безопасности. Некоторые из этих хакеров были успешны в прошлом, однако разработчики WordPress быстро отслеживают уязвимости и выпускают обновления для их устранения. Получается, будто миллионы людей постоянно тестируют и исправляют ваше программное обеспечение.

Базовая архитектура WordPress делает применение этих обновлений быстрым и безболезненным процессом, который могут выполнять даже начинающие веб-

пользователи. Если вы хорошо знаете, как настроить WordPress и обновить его до последних версий, когда они станут доступны, WordPress станет гораздо более безопасной платформой для вашего сайта, чем все остальное. Мы обсудим безопасность более подробно в *главе 8*.

Стоимость

WordPress — бесплатный продукт. PHP бесплатен. MySQL бесплатен. Большинство плагинов тоже бесплатны.

Серверы и хостинг стоят денег, но в зависимости от того, насколько велико ваше веб-приложение и сколько трафика вы получаете, это может быть относительно недорого. Если вам требуются пользовательские функции, которых нет ни в одном из существующих плагинов, вам, возможно, придется заплатить разработчику за создание нового. Или, если вы сами разработчик, это будет стоить вам времени.

Ответы на некоторые распространенные критические мнения о WordPress

Некоторые высококвалифицированные критики WordPress могут сказать, что это не очень хорошая основа для создания веб-приложений и это вообще не фреймворк. При всем уважении к тем, кто придерживается подобных мнений, нужно объяснить, почему мы не согласны. Далее приведены некоторые распространенные критические замечания.

WordPress подходит только для блогов

Многие люди считают, что поскольку WordPress впервые был создан для ведения блога, он хорош только для этой цели.

Подобные заявления были верны несколько лет назад, но с тех пор WordPress внедрил мощную функциональность CMS, что делает его полезным для других сайтов, ориентированных на контент. WordPress в настоящее время является самой популярной из используемых CMS с долей рынка более 60%². На рис. 1.1 показан слайд из презентации Мэтта Мулленвега "Состояние WordPress" с конференции WordCamp San Francisco 2013. Перевернутая пирамида слева изображает WordPress 2006 года, где большая часть кода посвящена приложению блога, и есть немного кода CMS и платформы, на котором она держится. Правая пирамида представляет текущее состояние платформы WordPress, где большая часть кода находится в самой платформе, поверх которой расположен слой CMS, а поверх уровня CMS — приложение для блога. Сейчас WordPress является гораздо более устойчивой платформой, чем несколько лет назад.

² W3Tech (bit.ly/w3techs) регулярно проводит исследования по использованию различных систем управления контентом.



Рис. 1.1. Диаграммы из презентации Мэтта Мулленберга "Состояние WordPress" 2013 год
WordPress не всегда был таким устойчивым

С помощью API пользовательских типов записей можно настроить установку WordPress для поддержки других типов контента, кроме постов или страниц блога. Мы подробно рассмотрим это в *главе 5*.

WordPress предназначен только сайтов, ориентированных на контент

Подобно людям с позицией "только для блогов", некоторые скажут, что WordPress предназначен только для управления контентом сайтов.

Во-первых, даже если бы WordPress был применим только к контентным сайтам и приложениям, на него приходилось бы большое количество приложений. Главный экран вашего телефона, вероятно, содержит множество приложений, основанных на контенте, таких как Netflix, Twitter, Facebook, Reddit и Evernote. Это очень популярные приложения, поддерживаемые гигантскими компаниями. Сейчас мы не говорим, что эти приложения работают на WordPress, но мы предполагаем, что *можно* создать приложение, похожее на это, с использованием WordPress в качестве фреймворка.

Во-вторых, как мы подробно рассмотрим в этой книге, WordPress — это отличная платформа для создания более интерактивных веб-приложений. Основной функцией, позволяющей выбрать WordPress в качестве основы, является API плагинов, который позволяет вам понять, как работает WordPress по умолчанию, и что-то изменить. Вам доступны не только тысячи плагинов в репозитории WordPress и других местах в Интернете, API плагинов позволяет вам написать собственные пользовательские плагины, чтобы WordPress делал что угодно с помощью PHP/MySQL.

WordPress не масштабируется

Некоторые из людей, которые так говорят, будут указывать на установку WordPress по умолчанию, работающую на хостинге нижнего уровня, и они отмечают, как сайт замедляется или "падает" при большой нагрузке, и, таким образом, приходят к выводу, что WordPress не масштабируется.

Или, может быть, когда мы предложили вам создать сайт, такой как Facebook, с использованием WordPress, вы справедливо насмеялись над этой идеей.



Если вы намереваетесь создать приложение в масштабе Facebook, эта книга не для вас. Спросите своего технического директора, какая часть их бюджета в миллиард долларов выделена вашему приложению и каких инженеров вам нужно переманить из Google и Amazon, чтобы создать собственное решение.

В действительности многие сайты с большим трафиком работают на WordPress. WordPress.com работает на том же базовом программном обеспечении, что и любой сайт WordPress, и является одним из самых популярных сайтов в мире.

По мере расширения вашего приложения вам необходимо будет обновлять и заменять отдельные компоненты, чтобы соответствовать новому масштабу. Проблемы с масштабированием WordPress такие же, как и при масштабировании любого приложения: кэширование страниц и данных, более быстрая обработка вызовов базы данных и повышение производительности сети. Крупные сайты, такие как WordPress.com, TechCrunch и блог *New York Times*, стали использовать WordPress. Точно так же большинство уроков по масштабированию приложений PHP/MySQL в целом применимы и к WordPress. Мы подробно расскажем о масштабировании приложений WordPress в *главе 14*.

WordPress небезопасен

Как и для любого продукта с открытым исходным кодом, при использовании WordPress есть компромисс в отношении безопасности.

С одной стороны, поскольку WordPress очень популярен, он часто становится целью хакеров, ищущих уязвимости безопасности. А поскольку исходный код открытый, уязвимости будет легче обнаружить.

С другой стороны, поскольку WordPress является продуктом с открытым исходным кодом, вы узнаете, если эксплойты станут общедоступными, и кто-то другой, вероятно, выпустит исправление для вас.

Мы чувствуем себя более уверенно, зная, что есть множество людей, пытающихся использовать WordPress, и столько же людей, которые работают над тем, чтобы защитить WordPress от взломов. Мы не верим в "безопасность через неизвестность", если только в качестве дополнительной меры. Мы бы предпочли, чтобы дыры в безопасности нашего программного обеспечения появлялись открыто, а не оставались незамеченными до самого худшего момента.

В *главе 8* более детально рассматриваются вопросы безопасности, в том числе дан список рекомендаций по повышению безопасности установки WordPress и описаны способы безопасной разработки.

Плагины WordPress ужасны

API плагинов в WordPress и тысячи плагинов, которые были разработаны с его использованием, являются "секретным соусом" и, по нашему мнению, основной причиной того, что WordPress стал настолько популярным и настолько успешным в качестве веб-платформы.

Некоторые люди скажут: "Конечно, есть тысячи плагинов, но они все ужасны". Ну да, некоторые из плагинов действительно не очень полезны.

Но есть много плагинов, которые определенно не являются чепухой, например AppPresser, разработанный одним из авторов этой книги Брайаном Мессенленером. Если вам нравится WordPress для управления своим письменным контентом или

интернет-магазином, то плагин и платформа AppPresser — вот самый быстрый способ получить данный контент или сохранить в мобильном приложении.

Плагин Paid Memberships Pro, разработанный другим автором Джейсоном Коулманом, тоже хорош. Использование Paid Memberships Pro для управления биллингом и участниками позволит вам сосредоточить усилия по разработке на основной функциональности вашего приложения, а не на том, как интегрировать ваш сайт с платежной системой.

Многие плагины делают что-то очень простое (например, скрывают панель администратора от обычных пользователей), работают именно так, как рекламируется, и на самом деле вовсе не ужасны.

Темы и плагины, найденные в репозитории WordPress.org, тщательно проверяются сообществом на предмет безопасности и качества кода. Общеизвестно, что обзор на темы WordPress.org (oreil.ly/WgTyD) более строгий и всеобъемлющий, чем в других платформах. Проект Tide (oreil.ly/iR94e) работает над добавлением автоматических тестов для плагинов и репозитория тем, что приведет к увеличению качества плагинов и обновлений, а также позволит обнаружить проблемы совместимости и безопасности быстрее.

Даже дрянные плагины можно исправить, переписать или заимствовать для улучшения их работы. Иногда проще переписать плохой плагин, чем исправлять его. Однако в данном случае вы окажетесь все же на шаг ближе к цели, чем при написании с нуля.

Никто не заставляет вас использовать плагины WordPress, не проверяя их самостоятельно. Если вы создаете серьезное веб-приложение, то должны самостоятельно проверить код плагина, исправить его в соответствии со своими стандартами и продолжить разработку.

Когда не следует использовать WordPress

WordPress не является универсальным решением для любого приложения. В этом разделе описываются несколько случаев, в которых вы *не хотели бы* использовать WordPress для создания своего приложения.

Вы планируете лицензировать или продавать технологию своего сайта

WordPress использует Общедоступную лицензию GNU версии 2 (GPLv2), которая содержит ограничения на то, как вы распространяете программное обеспечение, созданное с его помощью. А именно, вы не можете ограничивать то, что люди делают с вашим программным обеспечением, когда вы продаете или распространяете его.

Это сложная тема, но основная идея заключается в том, что если вы продаете только свое приложение или предоставляете *доступ* к нему, то вам не нужно беспоко-

иться о GPLv2. Однако если вы продаете или распространяете исходный код вашего приложения, то GPLv2 будет применяться к распространяемому вами коду.

Например, если мы размещаем SchoolPress на наших собственных серверах и продаем учетные записи для доступа к приложению, это не считается распространением, а GPLv2 никак не влияет на наш бизнес.

Но если мы захотим разрешить школам устанавливать программное обеспечение для запуска на своих собственных серверах, то нам необходимо предоставить им исходный код. Это будет считаться актом распространения. Наши клиенты смогут на законных основаниях бесплатно раздавать наш исходный код, даже если мы изначально взяли с них плату за программное обеспечение. Мы должны учитывать лицензию GPLv2, которая не позволяет нам ограничивать то, что пользователи делают с кодом после его загрузки.

Имеется другая платформа, которая приведет вас к цели быстрее

Если у вас есть команда опытных разработчиков Ruby, то целесообразно создавать веб-приложение именно на Ruby. Если есть платформа, фреймворк или пакет, включающий 80% функций Ruby, необходимых для вашего веб-приложения, и WordPress не имеет ничего подобного, то вам, вероятно, следует предпочесть эту другую платформу.

Гибкость не важна для вас

Одна из главных особенностей сайта WordPress — возможность быстрой замены частей сайта в соответствии с вашими потребностями. Например, если "лайки" Facebook перестали приносить трафик, вы можете удалить плагин Facebook Connect и установить плагин для Pinterest.

Как правило, обновление вашей темы или замена плагинов на сайте WordPress будет быстрее, чем разработка функций с нуля на другой платформе. Однако в тех случаях, когда оптимизация и производительность важнее, чем возможность быстрого обновления приложения, лучшим выбором будет программирование нативного приложения или программирование прямо на PHP.

Если ваше приложение служит для выполнения *одной простой задачи*, то вы, скорее всего, захотите построить его на более низком уровне. Например, сервер лицензий Paid Memberships Pro представляет собой один файл JSON с дополнительной информацией и небольшим скриптом для проверки лицензионных ключей и доставки сжатых файлов. Джейсон создал этот сервер лицензий на PHP с большим объемом кэширования. Сервер лицензий функционирует на DigitalOcean Sproplet за 10 долларов в месяц и обслуживает более 80 тыс. сайтов, работающих под управлением Paid Memberships Pro.

Точно так же, если у вас есть ресурсы масштаба Facebook, то вы можете позволить себе создавать все вручную и использовать собственные компиляторы PHP-to-C и

нативные компоненты iOS, чтобы сократить время загрузки вашего веб-сайта и приложения на несколько миллисекунд.

Ваше приложение должно работать в режиме реального времени

Одним из потенциальных недостатков WordPress, о котором мы поговорим позже, является его зависимость от типичной архитектуры веб-сервера. В стандартной настройке WordPress пользователь посещает URL-адрес, который связывается с веб-сервером (например, Apache) по HTTP, запускает скрипт PHP для генерации страницы, а затем возвращает пользователю полную страницу.

Существуют способы улучшить производительность этой архитектуры с использованием методов кэширования и/или оптимизированных настроек сервера. Вы можете сделать WordPress асинхронным с помощью вызовов Ajax или доступа к базе данных с помощью альтернативных клиентов. Однако если ваше приложение должно работать в режиме реального времени и быть полностью асинхронным (например, приложение, похожее на чат, или многопользовательская игра), мы советуем хорошенько подумать, подойдет ли вам WordPress.

Многие разработчики WordPress, включая Мэтта Мулленвега, основателя и духовного лидера WordPress, понимают это ограничение. Все больше функциональности WordPress переносится в JavaScript, где вычисления могут быть перенесены в браузер, а фреймворки, такие как REACT, позволяют создавать высокоинтерактивные события. Новый редактор Gutenberg, добавленный в WordPress 5.0, является лучшим примером этого шага и свидетельствует о грядущих событиях. Но в данный момент вы столкнетесь с трудностями, пытаясь заставить WordPress работать асинхронно с той же производительностью, что и нативное приложение или что-то полностью построенное на Node.js либо других технологиях, специально предназначенных для приложений реального времени.

WordPress как фреймворк

Системы управления контентом, такие как WordPress, Drupal и Joomla, часто остаются вне обсуждения фреймворков, но на самом деле WordPress (в частности) действительно отлично подходит для того, для чего предназначены фреймворки: для быстрого создания приложений.

В течение нескольких минут вы можете настроить WordPress и получить полнофункциональное приложение с регистрацией пользователей, управлением сеансами, управлением контентом и панелью для мониторинга активности сайта.

Различные API, общие объекты и вспомогательные функции, описанные в этой книге, позволяют быстрее программировать сложные приложения, не беспокоясь об интеграции систем более низкого уровня.

На рис. 1.2 показан правый треугольник из презентации Мэтта Мулленвега "Состояние WordPress" 2013 года, изображающий устойчивую платформу WordPress со

слоем CMS, находящимся сверху, и приложение для ведения блогов, построенное поверх слоя CMS.



Рис. 1.2. Платформа WordPress

Реальность такова, что большая часть текущей кодовой базы WordPress поддерживает лежащую в основе платформу приложений. Думайте о каждом выпуске WordPress как о фреймворке приложения, который поставляется с образцом приложения для блога.

WordPress и фреймворки Framework-View-Controller

Модель-представление-контроллер (MVC — Model-View-Controller) — это общий шаблон проектирования, присутствующий во многих средах разработки программного обеспечения. Основные преимущества архитектуры MVC — повторное использование кода и разделение задач (SoC — Separation of Concerns). WordPress не задействует архитектуру MVC, но по-своему реализует повторное использование кода и SoC.

Здесь мы кратко расскажем об архитектуре MVC и о том, как она влияет на процесс разработки в WordPress. Если вы знакомы с основами MVC, этот раздел должен помочь вам понять, как подходить к разработке на WordPress аналогичным образом. На рис. 1.3 приведено типичное приложение на основе MVC. Конечный пользователь использует *контроллер*, манипулирующий состоянием приложения и данными через *модель*, которая затем обновляет *представление*, отображаемое пользователю. Например, в приложении блога пользователь может просматривать страницу последних сообщений (представление). Пользователь щелкает заголовок сообщения, осуществляя переход к новому URL (контроллеру), который загрузит данные поста (в модель) и отобразит один пост (другое представление).

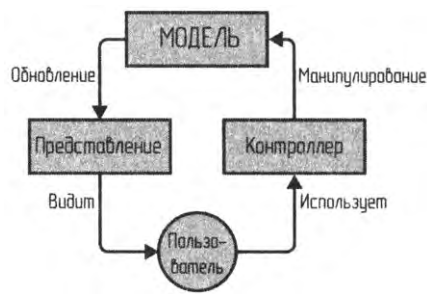


Рис. 1.3. Как работает MVC

Архитектура MVC поддерживает повторное использование кода, позволяя моделям, представлениям и контроллерам взаимодействовать. Например, и представление последних сообщений, и представление отдельных сообщений могут задействовать одну и ту же модель публикации при отображении данных публикации. Одни и те же модели могут служить в веб-интерфейсе для отображения сообщений и в бэкенде — для их редактирования. Архитектура MVC поддерживает SoC, позволяя дизайнерам сосредоточить свое внимание на представлениях, а программистам — на моделях.

Вы можете попробовать применить архитектуру MVC в WordPress. Есть целый ряд проектов, которые помогут вам сделать это. Однако мы считаем, что попытка привязать MVC к WordPress может привести к проблемам, если ядро WordPress не будет официально поддерживать MVC. Вместо этого мы предлагаем следовать "WordPress Way", как описано в этой книге.

Тем не менее, если вам интересно, WP MVC плагин (bit.ly/wp-mvc) находится в активной разработке и помогает вам на основе MVC Framework создавать плагины WordPress. Если вы не хотите или вам не нужен полный MVC, есть несколько способов привязать процесс MVC к WordPress.

Модели = плагины

В среде MVC код, который хранит базовые структуры данных и логику работы, находится в моделях. Именно здесь программисты будут проводить большую часть своего времени.

В WordPress плагины являются подходящим местом для хранения новых структур данных, сложной бизнес-логики и пользовательских определений типов записей.

Но это работает не всегда. Во-первых, многие плагины добавляют функциональность, подобную представлению, и содержат элементы дизайна (возьмите любой плагин, который добавляет виджет на ваши страницы). Во-вторых, формы и другие компоненты дизайна, используемые в инструментах WordPress, также обычно обрабатываются в плагинах.

Один из способов сделать SoC более понятным при добавлении в ваши плагины WordPress компонентов, отвечающих за дизайн, — создать папку `templates` или `pages` и поместить в нее свой код внешнего интерфейса. Обычная практика — позволить шаблонам (`templates`) переопределять шаблон, используемый плагином. Например, при работе WordPress с плагином `Paid Memberships Pro` вы можете скопировать папку с именем `paid-memberships-pro/pages` в папку с вашей активной темой, чтобы переопределить шаблоны страниц по умолчанию. (Этот метод переопределения шаблонов плагинов описан в *главе 4*.)

Представления = темы

В MVC-фреймворке код для отображения данных пользователю записывается в представлениях. Именно здесь дизайнеры и разработчики внешнего интерфейса будут проводить большую часть своего времени.

В WordPress темы — это подходящее место для хранения шаблонного кода и логики. Опять же, сравнение здесь не сопоставляется один к одному, но "views = themes" — хорошая отправная точка.

Контроллер = загрузчик шаблонов

В инфраструктуре MVC код для обработки пользовательского ввода (в виде URL-адресов или данных `$GET` или `$POST`) и определения того, какие модели и представления следует использовать для обработки запроса, хранится в контроллерах. Код контроллера обычно обрабатывается программистом и часто устанавливается один раз, а затем забывается. Программирование в приложении MVC происходит, по сути, в моделях и представлениях. Несмотря на это, контроллеры являются важной частью работы приложения.

В WordPress все запросы к страницам (если пользователи не обращаются к кэшированному файлу `*.html`) обрабатываются с помощью файла `index.php` в соответствии с иерархией шаблонов. Загрузчик шаблонов определяет, какой файл в шаблоне следует использовать для отображения страницы конечному пользователю. Например, `search.php` — для отображения результатов поиска, `single.php` — для отображения одного сообщения и т. д.

Поведение по умолчанию можно дополнительно настроить с помощью API-интерфейса `WP_Rewrite` (описанного в *главе 7*) и других хуков и фильтров. Информацию об иерархии шаблонов (bit.ly/temp-hier) вы можете найти в справочнике по WordPress. Мы более подробно рассмотрим иерархию шаблонов в *главе 4*.

Для лучшего понимания того, как работают фреймворки MVC, фреймворк PHP Yii (bit.ly/yii-guide) содержит большой ресурс, объясняющий подробно, как лучше всего использовать его архитектуру MVC³.

Подробнее о том, как разрабатывать веб-приложения на основе WordPress, читайте далее в этой книге.

Анатомия приложения WordPress

В данном разделе опишем приложение, которое мы создали как дополнение к этой книге: SchoolPress. Мы расскажем о предполагаемой функциональности SchoolPress, о том, как оно работает, и кто будет им пользоваться, и, что наиболее важно для этой книги, о том, как каждая часть приложения построена в WordPress.

Не беспокойтесь, если вы не понимаете некоторые из следующих терминов. В последующих главах мы рассмотрим всю введенную здесь терминологию более подробно. Когда это возможно, мы указываем на главу, которая соответствует обсуждаемой функции.

³ Yii — PHP-фреймворк на основе MVC. Другие платформы PHP, такие как Laravel (laravel.com), более популярны среди разработчиков WordPress и сообщества PHP в целом, но документация, связанная с MVC, на сайте Yii особенно хорошо написана.



В этой книге не ставилась задача "Как заново создать приложение SchoolPress" и здесь вы не найдете пошагового руководства. Когда это имеет смысл, мы просто ссылаемся на SchoolPress в наших примерах кода в этой книге, поэтому вам не нужно тратить время на понимание контекста каждого отдельного примера.

Что такое SchoolPress?

SchoolPress — это веб-приложение, которое позволяет учителям легко общаться со своими учениками за пределами классной комнаты. Учителя могут создавать *классы* и приглашать в них своих учеников. В каждом классе есть форум для специальных обсуждений, а также более структурированная система для учителей, чтобы публиковать *задания* и заставлять учащихся выполнять свою работу.

Рабочее приложение можно найти на веб-сайте SchoolPress (schoolpress.me). Исходный код приложения размещен в репозитории SchoolPress GitHub ([oreil.ly/Tt7PO](https://github.com/oreil/Tt7PO)).

SchoolPress работает в многосайтовой сети WordPress

SchoolPress запускает многосайтовую версию WordPress. На главном сайте размещены бесплатные аккаунты, где учителя могут зарегистрироваться и начать управлять своими классами. В нем также есть вся маркетинговая информация для отдельных школьных сайтов в сети, включая страницу, на которой можно зарегистрироваться и проверить платную подписку.

Школы могут создать уникальный поддомен, в котором будут размещаться занятия для учителей. Эта установка предлагает более точный контроль и отчетность для всех классов школы. Подробности использования многосайтовой сети с WordPress можно найти в *главе 12*.

Бизнес-модель SchoolPress

SchoolPress использует плагины Paid Memberships Pro, PMPro Register Helper и PMPro Network для настройки процесса регистрации и приема платежей по кредитной карте при регистрации школ.

Школы могут приобрести собственный уникальный поддомен за ежегодную плату. Другие пользователи SchoolPress не платят за доступ. Когда школьные администраторы регистрируются, они могут указать название школы и сокращение для своего поддомена `<ourschool>.schoolpress.me`. Для них создается новый сетевой сайт, и им предоставляется доступ к оптимизированной версии панели управления WordPress их сайта.

Затем администратор школы приглашает учителей в систему. Учителя также могут запросить приглашение в школу, которое должно быть одобрено администрацией школы. Учителя могут приглашать учеников в классы, которые они создают. Сту-

денты могут также запросить приглашение в класс, которое должно быть одобрено преподавателем.

Учителя могут также бесплатно зарегистрироваться, чтобы создавать свои классы по адресу *schoolpress.me*. Страницы, размещенные на поддомене, могут показывать рекламу или использовать другие схемы монетизации. Подробности того, как построить электронную торговлю с помощью WordPress, обсуждаются в *главе 15*.

Уровни участия и роли пользователей

Учителя получают уровень членства учителя (через Paid Memberships Pro) и специальную роль под названием "Учитель", которая дает им доступ к созданию и редактированию своих классов, модерированию дискуссий на их форумах классов, а также созданию и управлению назначениями для своих классов.

Учителя не имеют доступа к панели управления WordPress. Они формируют классы и управляют ими и заданиями через формы внешнего интерфейса, созданные для этих целей.

Студентам предоставляется уровень участия "Студент" и роль "Подписчик" по умолчанию в WordPress. Учащиеся имеют доступ к просмотру и участвуют только в тех классах, в которые их приглашают учителя. Подробные сведения о пользовательских ролях и возможностях описаны в *главе 6* и *главе 15*, где рассматриваются уровни членства для управления доступом.

Классы — это группы BuddyPress

Когда учителя создают "классы", они в действительности создают *группы BuddyPress* и приглашают своих учеников в группу. Благодаря BuddyPress мы получаем форумы классов, личные сообщения и хороший способ организации наших пользователей.

Дискуссионные форумы класса работают на плагине bbPress. Новый форум создается для каждого класса, и BuddyPress управляет доступом к форумам. Подробности об управлении сторонними плагинами, такими как BuddyPress и bbPress, можно найти в *главе 3*.

Назначения — это CPT

Назначения — это CPT (пользовательский тип записи), который использует форму представления внешнего интерфейса для учителей, чтобы публиковать новые назначения. Задания похожи на стандартные записи в блоге WordPress, с заголовком, содержимым тела и вложенными файлами. Учитель, отправляющий задание, является автором сообщения.



WordPress имеет встроенные типы сообщений, такие как сообщения и страницы, и встроенные таксономии, такие как категории и теги. Для SchoolPress мы создадим собственные CPT и таксономии. Подробнее о создании пользовательских типов записей и таксономий изложено в *главе 5*.

Представления (подтип) СРТ для назначений

Студенты могут публиковать комментарии по заданию, а также свои официальные представления через другую форму на веб-интерфейсе.

Представления, как и задания, также являются СРТ. Представления связываются с назначениями, устанавливая в поле `post_parent` представления идентификатор задания, в которое оно было отправлено. Студенты могут публиковать текстовые материалы, а также добавлять одно или несколько приложений к заявке.

Семестры являются таксономией для класса СРТ

Пользовательская таксономия с именем *Semester* настроена для группы/класса СРТ. Администраторы школ могут добавлять новые семестры на свои сайты. Например, можно создать семестр "осень-2019", и учителя могут назначить этот семестр при создании своих классов. Затем студенты смогут легко просмотреть список всех классов осени 2019 года.

Департаменты являются таксономией для класса СРТ

Пользовательская таксономия с именем *Департамент* тоже настроена для группы/класса СРТ. Они также доступны в виде раскрывающегося списка для учителей при создании классов и позволяют ученикам просматривать список классов по департаментам.

SchoolPress имеет один основной пользовательский плагин

За кулисами пользовательские фрагменты приложения SchoolPress управляются из одного пользовательского плагина под названием SchoolPress. Это основной плагин, он включает в себя определения для различных СРТ, таксономий и ролей пользователей. Он также содержит код для настройки сторонних плагинов, используемых SchoolPress, таких как Paid Memberships Pro и BuddyPress.

Основной плагин также содержит классы для школьных администраторов, учителей и учащихся, которые расширяют класс `WP_User`, и классы для классов, заданий и представлений, которые расширяют класс `WP_Post`. Эти (PHP) классы позволяют организовать наш код в объектно-ориентированном стиле, который облегчает контроль над тем, как работают наши различные настройки, и позволяет расширять код в будущем. С этими классами интересно работать, пример приведен в листинге 1.1.

Листинг 1.1. Возможные события входа пользователя

```
if($class->isTeacher($current_user))
{
    // это учитель, показать ему учительский материал
    //...
}
```

```

elseif($class->isStudent($current_user))
{
    // это ученик в классе, показать ему студенческие материалы
    //...
}
elseif(is_user_logged_in())
{
    // не вошли в систему, перенаправить их в форму входа
    wp_redirect(wp_login_url(get_permalink($class->ID)));
    exit;
}
else
{
    // не являются членами этого класса, перенаправить на страницу приглашения
    wp_redirect($class->invite_url);
    exit;
}

```

Создание пользовательских плагинов описано в *главе 3*, а расширение класса WP_User — в *главе 6*.

В SchoolPress есть несколько других пользовательских плагинов

Иногда для конкретного приложения будет разработан небольшой программный код, который также окажется полезным для других проектов. Если код достаточно содержательный, чтобы он мог работать вне контекста текущего приложения и основного плагина, то его можно встроить в отдельный пользовательский плагин.

Примером может быть Force First и Last Name как плагин Display Name (oreil.ly/QVANC), который необходим для этого проекта. Он не требует запуска основного кода плагина и полезен для других сайтов WordPress вне контекста приложения SchoolPress. Мы создали отдельный плагин для этой функциональности и поддерживаем его в репозитории *WordPress.org*, чтобы другие могли применить его и получить от него выгоду.

SchoolPress использует тему Memberlite

Основной сайт *schoolpress.me* работает на настроенной дочерней теме Memberlite. Если администратор школы подписывается на премиальный поддомен, он может выбрать одну из множества дочерних тем Memberlite; он также может изменить любой из цветов, шрифтов и логотипов темы, чтобы лучше соответствовать бренду школы/класса. Кроме того, все темы имеют адаптивный дизайн, который гарантирует, что сайт будет хорошо смотреться на экранах мобильных устройств и планшетов, а также на экранах персональных компьютеров.

Код темы Memberlite (memberlitetheme.com) очень строго ограничен, он включает только то, что связано с отображением. Код темы, очевидно, включает HTML и CSS для макета сайта, но также содержит некоторую простую логику, которая интегрируется с основным плагином SchoolPress (как и предыдущий код ветвления). Однако любой фрагмент кода, который манипулирует пользовательскими типами записей или пользовательскими ролями или требует большого количества вычислений, передается плагину SchoolPress.

Теперь, когда мы определили, что такое приложение, обсудили, почему вы можете создать его с помощью WordPress, представили способ разделения задач в стиле WordPress и описали на высоком уровне наше приложение-пример SchoolPress, давайте углубимся в суть WordPress, выясним, что в него включено и как он работает.

Основы WordPress

WordPress был впервые разработан в 2003 году и создавался преимущественно как программное обеспечение для блогов. К выпуску версии 3.5 образ WordPress изменился с блога на универсальную CMS, и слово "блог" было фактически удалено из описания программного обеспечения и большинства мест в исходном коде. Сегодня WordPress превратился в крупнейшую веб-платформу и используется примерно на 30% всех веб-сайтов в Интернете. Этот факт впечатляет еще больше, если привести абсолютные цифры, — более полутора миллиардов интернет-сайтов работают на WordPress.

WordPress за эти годы приобрел большую популярность по нескольким причинам. Во-первых, WordPress — это программное обеспечение с открытым исходным кодом, в которое вовлечено целое сообщество людей, вкладывающих средства в его совершенствование и постоянно пишущих новый код для расширения его функциональности. Пользователи, разработчики и дизайнеры WordPress всегда думают о новых способах использования WordPress и создают плагины для этих новых функций, которые могут быть доступны для сообщества.

Еще одна причина, по которой WordPress оказался настолько успешным, заключается в том, что это чрезвычайно гибкая CMS, включающая в себя хуки и фильтры, так что разработчики плагинов и тем могут иметь практически полный контроль при создании различных веб-сайтов. Разработчики постоянно изучают новые инновационные возможности программного обеспечения, в том числе создание веб-приложений и мобильных приложений, о которых говорится в этой книге. Применение хуков и фильтров рассматривается далее в этой главе.



Возможно, вы знаете, как использовать WordPress, и уже установили последнюю версию. Но если вы работаете с WordPress впервые, зайдите на домашнюю страницу WordPress (wordpress.org), чтобы познакомиться с ним.

Структура каталогов WordPress

Давайте кратко рассмотрим на верхнем уровне папки и файлы, создающиеся при типичной установке WordPress.

Корневой каталог

В корневом каталоге есть несколько основных файлов WordPress. Если вы не собираетесь копаться в коде ядра WordPress в поисках используемых хуков или не пы-

таетесь узнать, как написаны определенные функции, единственный системный файл WordPress, который вам может когда-либо понадобиться, — это `wp-config.php`. Никогда, никогда, никогда, никогда¹ не меняйте никакие другие файлы в ядре WordPress. Вмешиваться в файлы ядра — плохая идея, потому что обновление до новой версии WordPress все равно отменит ваши изменения. Единственный каталог, с которым вам нужно взаимодействовать, — это `wp-content`, поскольку он содержит ваши плагины, темы и загруженные файлы.

Каждый раз, когда вы захотите изменить основной файл WordPress, подумайте дважды. Вероятно, есть хук или фильтр, который вы можете использовать для достижения той же цели. Если для того, что вам нужно, нет хука или фильтра, создайте его и попросите добавить в ядро. Основные разработчики WordPress очень отзывчивы по поводу добавления новых хуков и фильтров.

Еще один файл, который вам может потребоваться обновить в корневом каталоге WordPress, в зависимости от настроек и способа применения WordPress: файл `*.htaccess`. Это не основной файл WordPress, а файл Apache, с помощью которого WordPress обрабатывает конфигурацию каталогов, постоянных ссылок и перенаправлений. Этот файл не существует по умолчанию, он создается WordPress автоматически при первом определении структуры ваших постоянных ссылок (permalink). Изучите все параметры конфигурации `*.htaccess` на досуге на странице поддержки WordPress [`htaccess \(oreil.ly/ByhN5\)`](http://oreil.ly/ByhN5).

Перечислим каталоги WordPress.

- ♦ `/wp-admin` — этот каталог содержит основные каталоги и файлы для управления интерфейсом панели администратора WordPress. Ключевой файл в этом каталоге — `admin-ajax.php`, через который должны выполняться все запросы Ajax. Мы рассмотрим Ajax в *главе 9*.
- ♦ `/wp-includes` — содержит основные каталоги и файлы для различных функций WordPress. Мы настоятельно рекомендуем вам просмотреть структуру и код в этом каталоге, чтобы лучше понять внутреннюю работу WordPress.
- ♦ `/wp-content` — в этом каталоге пользователи и разработчики WordPress могут заставить WordPress делать что угодно. Он содержит подкаталоги для плагинов и тем, которые установлены на вашем веб-сайте, а также любые медиафайлы, которые вы загружаете на свой веб-сайт.

Каталог `wp-content` содержит несколько подкаталогов, как описано далее.

- `/wp-content/plugins` — в этом каталоге будет находиться любой плагин WordPress, установленный на вашем сайте WordPress. По умолчанию WordPress поставляется с плагинами Hello Dolly и Akismet.

Hello Dolly включен в качестве быстрого примера того, как настраивается основной плагин WordPress. Сам плагин просто отображает случайную строку из песни "Hello Dolly" в правом верхнем углу панели администратора.

¹ ...никогда, никогда, никогда...

Плагин Akismet помогает останавливать спам в комментариях, проверяя входящие комментарии в базе данных по адресу **akismet.com**. Этот плагин и сервис значительно уменьшают количество спам-комментариев, которые попадают на ваш сайт. Услуга Akismet бесплатна (или сами назовите цену) для личного пользования.

- `/wp-content/themes` — в этом каталоге будет находиться любая тема WordPress, установленная на вашем сайте WordPress. По умолчанию WordPress поставляется с несколькими стандартными темами, названными по названию года выпуска (Twenty Seventeen — 2017, Twenty Nineteen — 2019 и т. д.).
- `/wp-content/uploads` — как только вы начнете загружать любые фотографии или файлы в библиотеку мультимедиа, то увидите, что данный каталог заполняется этими загруженными файлами. Все загруженные медиафайлы хранятся в каталоге `uploads`. Некоторые плагины также создают подкаталоги в каталоге `uploads` для различных файлов, используемых или управляемых плагином.
- `/wp-content/mu-plugins`. В WordPress вы можете принудительно активизировать любой плагин, создав каталог `mu-plugins` внутри каталога `wp-content`. Этот каталог не существует, пока вы его не создадите. "Mu" означает "must use — необходимо использовать", и любой плагин, который вы вставите в папку `mu-plugins`, будет автоматически запускаться без необходимости активации вручную на странице администрирования плагинов. На самом деле вы даже не увидите его в списке MU-плагинов.

Обязательные плагины особенно полезны в многосайтовых установках WordPress, потому что вы можете подключить плагины, которые администраторы вашего сайта не смогут деактивировать.

Рекомендуется проверить папку `mu-plugins` на каждом существующем сайте, над которым вы начинаете работать, и посмотреть, есть ли в нем какие-либо плагины, и, если да, определить, что они делают. Много раз отлаживая проблему, мы задавались вопросом, почему происходило что-то необычное, даже несмотря на то, что мы отключили все активные плагины, а в итоге оказывалось, что за эту проблему ответственен один упущенный `mu-плагин`.

Структура базы данных WordPress

WordPress работает поверх базы данных MySQL и создает свои собственные таблицы для хранения данных и контента. Далее приведена схема базы данных, созданная установкой WordPress по умолчанию. Мы также включили некоторую основную информацию о встроенных функциях WordPress для взаимодействия с этими таблицами. Если вы поймете схему базы данных и освоите функции из списка в этой главе, то сможете добавлять любые данные в WordPress и извлекать их из него.



В названиях следующих таблиц по умолчанию задан префикс `wp_`. Вы можете изменить этот префикс во время установки WordPress. Таким образом названия таблиц вашей установки WordPress могут отличаться.

Таблица `wp_options`

В таблице `wp_options` находятся данные всего сайта. Здесь хранятся имя, описание и адрес электронной почты администратора, которые вы указали при обычной установке. Эта таблица также поставляется с несколькими записями, которые хранят различные настройки по умолчанию в WordPress. В табл. 2.1 приведена структура базы данных для таблицы `wp_options`.

Таблица 2.1. Структура базы данных для таблицы `wp_options`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
<code>option_id</code>	<code>bigint(20)</code>		No	None	<code>AUTO_INCREMENT</code>
<code>option_name</code>	<code>varchar(64)</code>	<code>utf8_general_ci</code>	No		
<code>option_value</code>	<code>longtext</code>	<code>utf8_general_ci</code>	No	None	
<code>autoload</code>	<code>varchar(20)</code>	<code>utf8_general_ci</code>	No	Yes	

Настройки приложений и плагинов WordPress обычно хранятся в таблице `wp_options`, для этого предусмотрены функции, определенные в следующем разделе. Настройки могут храниться в отдельных строках и иметь общий префикс для имен параметров. В большинстве случаев целесообразно записывать все параметры в один массив и сохранять их в одной строке в таблице `wp_options`.

Функции в каталоге `/wp-includes/option.php`

В каталоге `/wp-includes/option.php` присутствуют следующие функции:

- ◆ `add_option(string $option, mixed $value = '', string $deprecated = '', string|bool $autoload = 'yes')` — сначала проверяет, существует ли `option_name` перед вставкой новой строки:
 - `$option` — обязательная для заполнения строка `option_name` — имя параметра, который вы хотите добавить;
 - `$value` — необязательная переменная `option_value` смешанного типа — значение, которое вы хотите добавить. Если переданная переменная является массивом или объектом, то значение будет сериализовано перед сохранением в базе данных;
 - `$deprecated` — этот параметр устарел в версии 2.3, больше не используется²;
 - `$autoload` — необязательный логический тип, указывающий, загружать ли опцию в кэш при запуске WordPress, и имеющий значения `yes` или `no`. Значе-

² Третий параметр для `add_option`, который устарел в версии 2.3, раньше был строкой "описания", которая хранилась вместе с параметром в таблице `wp_options`.

ние по умолчанию — `yes`. Если вы уверены, что вам понадобится эта опция при каждой загрузке страницы, то можете оставить значение по умолчанию. Если эта опция нужна вам только на определенных страницах, то для автозагрузки лучше установить значение `no`.

◆ `update_option($option, $newvalue)` — обновляет существующую опцию, но также создает ее, если она еще не существует:

- `$option` — обязательная для заполнения строка `option_name` — имя параметра, который вы хотите добавить/обновить;
- `$newvalue` — необязательная переменная `option_value` смешанного типа — значение, которое вы хотите добавить/обновить.

◆ `get_option($option, $default = false)` — извлекает `option_value` для предоставленного `option_name`:

- `$option` — обязательная для заполнения строка `option_name` — имя параметра, который вы хотите получить;
- `$default` — необязательная переменная смешанного типа, которую вы хотели бы вернуть, если в таблице отсутствует `option_name`, которое вы указали. По умолчанию этот параметр имеет значение `false`.

◆ `delete_option($option)` — удаляет существующую опцию из базы данных навсегда:

- `$option` — обязательная для заполнения строка `option_name` — имя параметра, который вы хотите удалить.



Большинство примеров кода в этой книге — не полностью функциональный программный код, а основные примеры использования функций, о которых мы говорим. Вы можете выполнять большинство примеров программ в пользовательском плагине или в файле `functions.php` вашей темы.

В листинге 2.1 продемонстрированы некоторые основные функции для взаимодействия с таблицей `wp_options`.

Листинг 2.1. Добавление, обновление, получение и удаление записей в таблице `wp_options`

```
<?php
// добавить опцию
$twitters = array('@bwawwp', '@bmess', '@jason_coleman');
add_option('bwawwp_twitter_accounts', $twitters);

// получить значение опции
$bwawwp_twitter_accounts = get_option('bwawwp_twitter_accounts');
echo '<pre>';
print_r($bwawwp_twitter_accounts);
echo '</pre>';
```

```

// обновить опцию
$twitters = array_merge(
    $twitters,
    array(
        '@alphaweb',
        '@pmproplugin'
    )
);
update_option('bwawwp_twitter_accounts', $twitters);

// получить значение опции
$bawawwp_twitter_accounts = get_option('bwawwp_twitter_accounts');
echo '<pre>';
print_r($bwawwp_twitter_accounts);
echo '</pre>';

// удалить опцию
delete_option('bwawwp_twitter_accounts');

/*
Результат выполнения этого листинга должен выглядеть примерно так:
Array
(
    [0] => @bwawwp
    [1] => @bmess
    [2] => @jason_coleman
)
Array
(
    [0] => @bwawwp
    [1] => @bmess
    [2] => @jason_coleman
    [3] => @alphaweb
    [4] => @pmproplugin
)
*/
?>

```

Таблица `wp_users`

Когда вы входите в WordPress с вашим именем пользователя и паролем, то ссылаетесь на данные, хранящиеся в этой таблице. Все пользователи и их данные по умолчанию хранятся в таблице `wp_users`. В табл. 2.2 приведена структура базы данных для таблицы `wp_users`.

Таблица 2.2. Структура базы данных для таблицы `wp_users`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
ID	bigint (20)		No	None	AUTO_INCREMENT
user_login	varchar (60)	utf8_general_ci	No		
user_pass	varchar (64)	utf8_general_ci	No		
user_nicename	varchar (50)	utf8_general_ci	No		
user_email	varchar (100)	utf8_general_ci	No		
user_url	varchar (100)	utf8_general_ci	No		
user_registered	datetime		No	0000-00-00 00:00:00	
user_activation_key	varchar (60)	utf8_general_ci	No		
user_status	int (11)		No	0	
display_name	varchar (250)	utf8_general_ci	No		

Для многих приложений WordPress создавать и управлять пользователями вы будете через графический интерфейс администратора. Однако если вам потребуется создать пользователей в своем программном коде или обновить метаданные о них, то будут полезны функции, описанные в следующем разделе.

Функции в каталоге `/wp-includes/...`

Эти функции находятся в файлах `/wp-include/pluggable.php` и `/wp-includes/user.php`.

♦ `wp_insert_user($userdata)` — создает нового пользователя в базе данных. Эту функцию также можно задействовать для обновления пользователя, если идентификатор пользователя передается вместе с `$user_data`. `$userdata` — обязательный массив имен и значений полей. Поля, которые нужно передать, следующие:

- ID — целое число, которое будет использоваться для обновления существующего пользователя;
- `user_pass` — строка, содержащая простой текстовый пароль пользователя;
- `user_login` — строка, содержащая имя пользователя для входа в систему;
- `user_nicename` — строка, содержащая удобное для пользователя имя пользователя. По умолчанию совпадает с `username` пользователя;
- `user_url` — строка, содержащая URL-адрес веб-сайта пользователя;
- `user_email` — строка, содержащая адрес электронной почты пользователя;

- `display_name` — строка, которая будет показана на сайте. По умолчанию совпадает с именем пользователя. Вполне вероятно, что вы захотите изменить его, чтобы все отображалось красиво;
 - `nickname` — псевдоним пользователя. По умолчанию совпадает с `username` пользователя;
 - `first_name` — имя пользователя;
 - `last_name` — фамилия пользователя;
 - `description` — строка, содержащая информацию о пользователе;
 - `rich_editing` — строка активации визуального редактора. Значение `false`, если не пуста;
 - `user_registered` — дата, когда пользователь зарегистрировался. Формат: `Y-m-d H:i:s`;
 - `role` — строка, используемая для установки роли пользователя.
- ◆ `wp_create_user($username, $password, $email)` — эта функция использует предыдущую функцию `wp_insert_user()` и облегчает добавление нового пользователя на основе обязательных столбцов:
- `$username` — обязательная строка имени пользователя/логина нового пользователя;
 - `$password` — обязательная строка пароля нового пользователя;
 - `$email` — обязательная строка адреса электронной почты нового пользователя.
- ◆ `wp_update_user($userdata)` — функция для обновления любого поля в таблицах `wp_users` и `wp_usermeta` (рассматривается далее), связанных с конкретным пользователем. Обратите внимание, что если пароль пользователя обновляется, все его `cookie` будут удалены, произойдет выход из его аккаунта WordPress:
- `$userdata` — обязательный массив имен полей и значений. `ID` и хотя бы одно другое поле обязательны для заполнения. Эти поля те же, что приняты в функции `wp_insert_post()`.
- ◆ `get_user_by($field, $value)` — эта функция возвращает объект `WP_User` в случае успеха и `false` — в противном случае. Класс WordPress `User` находится в файле `/wp-includes/capabilities.php` и в основном делает запросы в таблицу `wp_user` следующим образом: `SELECT * FROM wp_users WHERE $field = $value`.
- Класс `WP_User` кэширует результаты, чтобы избежать запросов к базе данных каждый раз, когда он вызывается. Класс также использует роли и возможности конкретного пользователя, о чем мы более подробно расскажем в *главе 6*:
- `$field` — обязательная строка поля, по которой вы хотите запросить данные пользователя. Эта строка может быть только `id`, `slug`, `email` или `login`;
 - `$value` — обязательное целое число или строка — значение для данного идентификатора, описания, адреса электронной почты или имени.

- ◆ `get_userdata($userid)` — эта функция фактически вызывает предыдущую функцию `get_user_by()` и возвращает тот же объект `WP_User`:
 - `$userid` — обязательное целое число — идентификатор пользователя, для которого вы хотите получить данные.
- ◆ `wp_delete_user($id, $reassign = 'novalue')` — эта функция удаляет пользователя, а также может переназначать любые его сообщения или ссылки другому пользователю:
 - `$id` — обязательное целое число — идентификатор пользователя, которого вы хотите удалить;
 - `$reassign` — необязательное целое число — идентификатор пользователя, на которого вы хотите переназначить любой пост или ссылку удаленного пользователя.

В листинге 2.2 продемонстрированы некоторые основные функции для взаимодействия с таблицей `wp_users`.

Листинг 2.2. Работа с таблицей `wp_users`

```
<?php
// создать пользователя в БД
$userdata = array(
    'user_login'    => 'brian',
    'user_pass'     => 'K003gT7@n*',
    'user_nicename' => 'Brian',
    'user_url'      => 'https://alphaweb.com/',
    'user_email'    => 'brian@alphaweb.com',
    'display_name'  => 'Brian',
    'nickname'      => 'Brian',
    'first_name'    => 'Brian',
    'last_name'     => 'Messenlehner',
    'description'   => 'This is a WordPress Administrator account.',
    'role'          => 'administrator'
);
wp_insert_user($userdata);

// создать пользователей
wp_create_user('jason', 'YR529G*v@', 'jason@schoolpress.me');

// получить пользователя по логину
$user = get_user_by('login', 'brian');
echo 'email: ' . $user->user_email . ' / ID: ' . $user->ID . '<br>';
echo 'Hi: ' . $user->first_name . ' ' . $user->last_name . '<br>';

// получить пользователя по адресу электронной почты
$user = get_user_by('email', 'jason@schoolpress.me');
echo 'username: ' . $user->user_login . ' / ID: ' . $user->ID . '<br>';
```

```
// обновляем пользователя, меняем поля имени пользователя
// и меняем роль на администратора
$userdata = array(
    'ID'           => $user->ID,
    'first_name'   => 'Jason',
    'last_name'    => 'Coleman',
    'user_url'     => 'http://strangerstudios.com/',
    'role'         => 'administrator'
);
wp_update_user($userdata);

// получаем данные пользователя для пользователя с именем Brian
$user = get_userdata($user->ID);
echo 'Hi: ' . $user->first_name . ' ' . $user->last_name . '<br>';

// удаляем пользователя - удаляем оригинального администратора
// и присваиваем его посты нашему новому администратору
// wp_delete_user(1, $user->ID);
/*
Результат выполнения приведенного листинга должен выглядеть примерно так:
email: brian@schoolpress.me / ID: 2
Hi: Brian Messenlehner
username: jason / ID: 3
Hi: Jason Coleman
*/
?>
```

Таблица wp_usermeta

Иногда вы можете захотеть сохранить дополнительные данные о пользователе. WordPress предлагает простой способ сделать это без добавления дополнительных столбцов в таблицу пользователей. Вы можете хранить столько метаданных пользователя, сколько вам нужно, в таблице wp_usermeta. Каждая запись связана с идентификатором пользователя в таблице wp_user полем user_id. В табл. 2.3 показана структура базы данных для таблицы wp_usermeta.

Таблица 2.3. Структура базы данных для таблицы wp_usermeta

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
umeta_id	bigint(20)		No	None	AUTO_INCREMENT
user_id	bigint(20)		No	0	
meta_key	varchar(255)	utf8_general_ci	Yes	NULL	
meta_value	longtext	utf8_general_ci	Yes	NULL	

Функции в wp_usermeta

◆ `get_user_meta($user_id, $key = '', $single = false)` — получает метазначение пользователя для указанного ключа:

- `$user_id` — обязательное целое число — идентификатор пользователя;
- `$key` — необязательная строка метаключа значения, которое вы хотите вернуть. Если пусто, будут возвращены все метаданные для данного пользователя;
- `$single` — значение логического типа, возвращать ли единственное значение или нет. Значение по умолчанию — `false`, т. е. значение будет возвращено в виде массива.

Для одного и того же идентификатора пользователя может быть несколько метаключей с разными значениями. Если вы установите `$single` в значение `true`, вы получите значение первого ключа; если вы установите значение `false`, вы получите массив значений каждой записи с тем же ключом.

◆ `update_user_meta($user_id, $meta_key, $meta_value, $prev_value = '')` — эта функция обновит метаданные пользователя и при этом создаст метаданные, если переданный ключ еще не существует:

- `$user_id` — обязательное целое число — идентификатор пользователя;
- `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите сохранить. Если этот метаключ уже существует, он обновит метазначение и присвоит ему текущую строку; если нет, функция создаст новую запись;
- `$meta_value` — обязательное значение смешанного типа — целое число, строка, массив или объект. Массивы и объекты будут автоматически сериализованы;
- `$prev_value` — необязательное значение смешанного типа — текущее значение метаданных. Если совпадение найдено, оно заменит предыдущее/текущее значение на указанное вами новое значение. Если оставить пустым, то новое метазначение заменит первый экземпляр соответствующего ключа. Если у вас есть пять строк метаданных с одним и тем же ключом, и вы не указываете, какую строку из них обновлять с помощью этого значения, то будет обновлена первая строка и удалены остальные четыре.

Примечание

Эта функция опирается на функцию `update_metadata()`, расположенную в каталоге `/wp-includes/meta.php`. Обязательно убедитесь в этом!

◆ `add_user_meta($user_id, $meta_key, $meta_value, $unique = false)` — эта функция будет вставлять совершенно новую метайнформацию пользователя в таблицу `wp_usermeta`. Мы больше не используем эту функцию, потому что мы можем просто вызвать `update_user_meta()` для вставки новых строк, а также их обновления. Если вы хотите убедиться, что данный метаключ используется только один раз

для каждого пользователя, вам следует использовать эту функцию и установить для параметра `$unique` значение `true`:

- `$user_id` — обязательное целое число — идентификатор пользователя;
- `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите сохранить;
- `$meta_value` — обязательное значение смешанного типа – целое число, строка, массив или объект;
- `$unique` — необязательный параметр логического типа, который при значении `true` гарантирует, что метаключ может быть добавлен только один раз для данного идентификатора.

◆ `delete_user_meta($user_id, $meta_key, $meta_value = '')` — удаляет метаданные пользователя для предоставленного идентификатора пользователя и соответствующего ключа. Вы также можете указать соответствующее метазначение, если хотите удалить только это значение, а не другие строки метаданных с тем же метаключом:

- `$user_id` — обязательное целое число — идентификатор пользователя;
- `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите удалить;
- `$meta_value` — необязательное значение смешанного типа — текущее значение метаданных. Если у вас есть несколько записей с одним и тем же метаключом, вы можете указать, какую из них удалить, сопоставив метазначение. По умолчанию значение пусто, что приведет к удалению всех метастроков, где найдено соответствие, включая `user_id` и `meta_key`.

В листинге 2.3 продемонстрированы некоторые основные функции для взаимодействия с таблицей `wp_username`.

Листинг 2.3. Работа с таблицей `wp_username`

```
<?php
// получить идентификатор Брайана
$brian_id = get_user_by('login', 'brian')->ID;

// добавляем пользовательские метаданные, $unique установлен в true
add_user_meta($brian_id, 'bwawwp_wife', 'Married to the game', true);

// получить пользовательские метаданные - вернуть одно значение
$brians_wife = get_user_meta($brian_id, 'bwawwp_wife', true);
echo "Brian's wife: " . $brians_wife . "<br>";

// добавляем пользовательские метаданные - 3-й параметр является уникальным
add_user_meta($brian_id, 'bwawwp_kid', 'Dalya');
add_user_meta($brian_id, 'bwawwp_kid', 'Brian');
```



```

add_user_meta($brian_id, 'bwawwp_kid', 'Nina');
add_user_meta($brian_id, 'bwawwp_kid', 'Cam');
add_user_meta($brian_id, 'bwawwp_kid', 'Aksel');

// обновить пользовательские метаданные - это обновит brian до brian jr.
update_user_meta($brian_id, 'bwawwp_kid', 'Brian Jr', 'Brian');

// получаем метаданные пользователя - возвращаем массив
$brians_kids = get_user_meta($brian_id, 'bwawwp_kid');
echo "Brian's kids:";
echo '<pre>';
print_r($brians_kids);
echo '</pre>';

// удаляем метаданные пользователя Брайана
delete_user_meta($brian_id, 'bwawwp_wife');
delete_user_meta($brian_id, 'bwawwp_kid');

// получить идентификатор пользователя Jason
$jason_id = get_user_by('login', 'jason')->ID;

// обновить пользовательские метаданные - это создаст их,
// если ключ не существует для пользователя
update_user_meta($jason_id, 'bwawwp_wife', 'Kimberly Ann Coleman');

// получаем метаданные пользователя - возвращаем массив
$jasons_wife = get_user_meta($jason_id, 'bwawwp_wife');
echo "Jason's wife:";
echo '<pre>';
print_r($jasons_wife);
echo '</pre>';

// добавляем метаданные, хранящие данные пользователя в виде массива
add_user_meta($jason_id, 'bwawwp_kid', array('Isaac', 'Marin'));

// получаем метаданные пользователя, возвращая единственное значение,
// которое оказывается массивом
$jasons_kids = get_user_meta($jason_id, 'bwawwp_kid', true);
echo "Jason's kids:";
echo '<pre>';
print_r($jasons_kids);
echo '</pre>';

// удаляем метаданные пользователя Джейсон
delete_user_meta($jason_id, 'bwawwp_wife');
delete_user_meta($jason_id, 'bwawwp_kid');

```

```
/*
Результат выполнения этого листинга должен выглядеть примерно так:
Brian's wife: Married to the game
Brian's kids:
Array
(
    [0] => Dalya
    [1] => Brian Jr
    [2] => Nina
    [3] => Cam
    [4] => Aksel
)
Jason's wife:
Array
(
    [0] => Kimberly Ann Coleman
)
Jason's kids:
Array
(
    [0] => Isaac
    [1] => Marin
)
*/
?>
```

Таблица wp_posts

Ах, WordPress. В таблице wp_posts хранится большинство ваших данных постов. По умолчанию WordPress поставляется с постами и страницами. И посты, и страницы технически являются постами и хранятся в этой таблице. Данные в поле post_type позволяют различить тип сообщения, т. е. выяснить, является ли оно сообщением, страницей, пунктом меню, комментарием или любым другим СРТ, который вы можете создать позже (СРТ более подробно рассматривается в главе 5). В табл. 2.4 показана структура базы данных для таблицы wp_posts.

Таблица 2.4. Структура базы данных для таблицы wp_posts

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
ID	bigint(20)		No	None	AUTO_INCREMENT
post_author	bigint(20)		No	0	
post_date	datetime		No	0000-00-00 00:00:00	

Таблица 2.4 (окончание)

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
post_date_gmt	datetime		No	0000-00-00 00:00:00	
post_content	longtext	utf8_general_ci	No	None	
post_title	text	utf8_general_ci	No	None	
post_excerpt	text	utf8_general_ci	No	None	
post_status	varchar(20)	utf8_general_ci	No	Publish	
comment_status	varchar(20)	utf8_general_ci	No	Open	
ping_status	varchar(20)	utf8_general_ci	No	Open	
post_password	varchar(20)	utf8_general_ci	No		
post_name	varchar(200)	utf8_general_ci	No		
to_ping	text	utf8_general_ci	No	None	
pinged	text	utf8_general_ci	No	None	
post_modified	datetime		No	0000-00-00 00:00:00	
post_modified_gmt	datetime		No	0000-00-00 00:00:00	
post_content_filtered	longtext	utf8_general_ci	No	None	
post_parent	bigint(20)		No	0	
guid	varchar(255)	utf8_general_ci	No		
menu_order	int(11)		No	0	
post_type	varchar(20)	utf8_general_ci	No	Post	
post_mime_type	varchar(100)	utf8_general_ci	No		
comment_count	bigint(20)		No	0	

Функции в каталоге /wp-includes/post.php

Следующие функции находятся в файле post.php каталога /wp-includes/:

- ◆ `wp_insert_post($postarr, $wp_error = false)` — эта функция вставляет новое сообщение с указанными данными:
 - `$postarr` — массив или объект данных поста. Ожидается, что массивы будут экранированы, а объекты — нет;
 - `$wp_error` — необязательный параметр логического типа, который допускает `WP_Error`, если возвращается `false`.

Значения по умолчанию для параметра `$postarr`:

- `post_status` — по умолчанию черновик (`draft`);
- `post_type` — по умолчанию пост (`post`);
- `post_author` — по умолчанию используется текущий идентификатор пользователя (`$user_ID`), который добавил пост;
- `ping_status` — по умолчанию это значение в опции `default_ping_status`. Может ли приложение принимать пинги;
- `post_parent` — по умолчанию 0. Установите в это значение идентификатор поста, которому с которым данный пост связан, если таковой имеется;
- `menu_order` — по умолчанию 0. Порядок, в котором отображается массив;
- `to_ping` — нужно ли пинговать;
- `pinged` — по умолчанию пустая строка;
- `post_password` — по умолчанию пустая строка. Пароль для доступа к вложению;
- `guid` — глобальный уникальный идентификатор для ссылки на вложение;
- `post_content_filtered` — содержание публикации отфильтровано;
- `post_excerpt` — аннотация поста.

◆ `wp_update_post($postarr = array(), $wp_error = false)` — эта функция обновляет пост указанными данными:

- `$postarr` — обязательный массив или объект данных поста. Ожидается, что массивы будут экранированы, а объекты — нет;
- `$wp_error` — необязательный параметр логического типа, который допускает `WP_Error`, если возвращается `false`.

◆ `get_post($post = null, $output = OBJECT, $filter = 'raw')` — эта функция получает данные поста из предоставленного идентификатора поста или объекта поста:

- `$post` — необязательное целое число или объект — идентификатор записи или объекта записи, который вы хотите получить. По умолчанию используется текущее сообщение, в котором вы находитесь, внутри цикла публикаций. Его обсудим далее в этой главе;
- `$output` — необязательная строка формата вывода. Значением по умолчанию является `OBJECT` (объект `WP_Post`), а другими значениями могут быть `ARRAY_A` (ассоциативный массив) или `ARRAY_N` (числовой массив);
- `$filter` — необязательная строка, сообщающая о том, как контекст должен быть очищен при выводе. Значение по умолчанию — `raw`, другие возможные значения — `edit`, `db`, `display`, `attribute` или `js`. Эта тема описана в *главе 8*.

◆ `get_posts($args = null)` — эта функция возвращает список сообщений, соответствующих критериям. Функция использует класс `WP_Query`, примеры которого вы увидите далее в книге.

`$args` — необязательный массив аргументов поста. Значения по умолчанию:

- `numberposts` — по умолчанию 5. Общее количество сообщений, которые необходимо получить (значение `-1` — все сообщения);
- `offset` — по умолчанию 0. Число записей, которые необходимо пропустить;
- `category` — из какой категории вернуть посты;
- `orderby` — по умолчанию это `post_date`. Как сортировать посты;
- `order` — по умолчанию `DESC` (по убыванию) Порядок получения сообщений;
- `include` — список идентификаторов сообщений, которые нужно вернуть;
- `exclude` — список идентификаторов постов, которые надо исключить;
- `meta_key` — произвольный ключ метаданных;
- `meta_value` — произвольное значение метаданных. Также необходимо использовать `meta_key`;
- `post_type` — по умолчанию `post` (`post`). Может быть страница (`page`), или вложение (`attachment`), или любой пользовательский СРТ. Строка `any` будет возвращать сообщения всех типов;
- `post_parent` — идентификатор родительского поста;
- `post_status` — по умолчанию опубликовано (`publish`). Статус сообщения для получения.

◆ `wp_delete_post($postid = 0, $force_delete = false)` — эта функция поместит любое сообщение в корзину или окончательно удалит его, если для `$force_delete` установлено значение `true`:

- `$postid` — обязательное целое число — идентификатор поста, который вы хотите удалить;
- `$force_delete` — необязательный логический тип, который, если задано значение `true`, удалит сообщение; если оставить это поле пустым, по умолчанию оно будет иметь значение `false` — переместить сообщение в корзину.

В листинге 2.4 продемонстрированы некоторые основные функции для взаимодействия с таблицей `wp_posts`.

Листинг 2.4. Работа с таблицей `wp_posts`

```
<?php
// вставить пост в БД, установить статус сообщения - черновик
$args = array(
    'post_title' => 'Building Web Apps with WordPress',
```

```

        'post_excerpt' => 'WordPress as an Application Framework',
        'post_content' => 'WordPress is the key to successful cost effective web
solutions in most situations. Build almost anything on top of the WordPress
platform. DO IT NOW!!!!',
        'post_status' => 'draft',
WordPress Database Structure | 39
        'post_type' => 'post',
        'post_author' => 1,
        'menu_order' => 0
    );
$post_id = wp_insert_post($args);
echo 'post ID: ' . $post_id . '<br>';

// обновить сообщение - изменить статус сообщения на "опубликовано"
$args = array(
    'ID' => $post_id,
    'post_status' => 'publish'
);
wp_update_post($args);

// получить пост - вернуть данные поста как объект
$post = get_post($post_id);
echo 'Object Title: ' . $post->post_title . '<br>';

// получить пост - вернуть данные поста как массив
$post = get_post($post_id, ARRAY_A);
echo 'Array Title: ' . $post['post_title'] . '<br>';

// удалить пост - пропустить корзину и удалить его навсегда
wp_delete_post($post_id, true);

// получить посты - вернуть 100 сообщений
$posts = get_posts(array('numberposts' => '100'));

// пройтись в цикле через все сообщения и отобразить идентификатор
// и заголовок для каждого
foreach ($posts as $post) {
    echo $post->ID . ': ' . $post->post_title . '<br>';
}

/*
Результат выполнения этого листинга должен выглядеть примерно так:
post ID: 589
Object Title: Building Web Apps with WordPress
Array Title: Building Web Apps with WordPress
"A list of post IDs and Titles from your install"
*/
?>

```

Таблица `wp_postmeta`

Иногда вы можете захотеть сохранить дополнительные данные о постах. WordPress предлагает простой способ сделать это без добавления дополнительных полей в таблицу постов. Вы можете хранить столько метаданных постов, сколько вам нужно, в таблице `wp_postmeta`. Каждая запись связана с постом через поле `post_id`. При редактировании любого сообщения на бэкенде WordPress вы можете добавлять/обновлять/удалять метаданные или настраиваемые поля через пользовательский интерфейс. В табл. 2.5 показана структура базы данных для таблицы `wp_postmeta`.



Ключи метаданных, начинающиеся с подчеркивания, скрыты в пользовательском интерфейсе и недоступны для редактирования пользовательских полей на странице редактирования поста. Это полезно, чтобы скрыть определенные метаполя, если вы не хотите, чтобы конечные пользователи редактировали такие поля напрямую.

Таблица 2.5. Структура базы данных для таблицы `wp_postmeta`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
<code>meta_id</code>	<code>bigint(20)</code>		No	None	<code>AUTO_INCREMENT</code>
<code>post_id</code>	<code>bigint(20)</code>		No	0	
<code>meta_key</code>	<code>varchar(255)</code>	<code>utf8_general_ci</code>	Yes	NULL	
<code>meta_value</code>	<code>longtext</code>	<code>utf8_general_ci</code>	Yes	NULL	

Функции из каталога `/wp-includes/post.php`

Следующие функции находятся в файле `/wp-includes/post.php`.

- ◆ `get_post_meta($post_id, $key = '', $single = false)` — получить метаданные сообщения для данного сообщения:
 - `$post_id` — обязательное целое число — идентификатор поста, для которого вы хотите получить метаданные;
 - `$key` — необязательная строка имени метаключа, для которой вы хотите получить метаданные. По умолчанию возвращаются метаданные для всех метаключей для определенного сообщения;
 - `$single` — значение логического типа, возвращать ли единственное значение или нет. По умолчанию — `false`, т. е. значение будет возвращено в виде массива.

Для одного и того же идентификатора поста может быть несколько метаключей с разными значениями. Если вы установите `$single` в значение `true`, то получите значение первого ключа; если вы установите значение `false`, то получите массив значений каждой записи с тем же ключом.

- ◆ `update_post_meta($post_id, $meta_key, $meta_value, $prev_value = '')` — эта функция обновит метаданные поста и создаст метаданные, если переданный ключ еще не существует:
 - `$post_id` — обязательное целое число идентификатора записи;
 - `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите сохранить. Если этот метаключ уже существует, он обновит метазначение и присвоит ему текущую строку; если нет, он вставит новую запись;
 - `$meta_value` — обязательное значение смешанного типа — целое число, строка, массив или объект. Массивы и объекты будут автоматически сериализованы;
 - `$prev_value` — необязательное значение смешанного типа — текущее значение метаданных. Если совпадение найдено, оно заменит предыдущее/текущее значение на указанное вами новое значение. Если оставить пустым, новое метазначение заменит первый экземпляр соответствующего ключа. Если у вас есть пять строк метаданных с одним и тем же ключом, и вы не указываете, какую строку обновлять с помощью этого значения, обновится первая строка и будут удалены остальные четыре.



Эта функция опирается на функцию `update_metadata()`, расположенную в `/wp-includes/meta.php`. Обязательно посмотрите!

- ◆ `add_post_meta($post_id, $meta_key, $meta_value, $unique = false)` — эта функция вставляет новую метазапись в таблицу `wp_postmeta`. Эта функция встречается реже, потому что мы можем просто вызвать предыдущую функцию, о которой мы говорили, `update_post_meta()`, чтобы вставлять новые строки и обновлять их. Если вы хотите убедиться, что данный метаключ используется только один раз на пост, вы должны вызвать эту функцию и установить для параметра `$unique` значение `true`:
 - `$user_id` — обязательное целое число — идентификатор записи;
 - `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите сохранить;
 - `$meta_value` — обязательное значение смешанного типа — целое число, строка, массив или объект;
 - `$unique` — необязательный параметр логического типа, который гарантирует, что метаключ может быть добавлен только один раз для данного идентификатора.
- ◆ `delete_post_meta($post_id, $meta_key, $meta_value = '')` — эта функция удаляет метаданные записи для предоставленного идентификатора записи и соответствующего ключа. Вы также можете указать соответствующее метазначение, если

хотите удалить только это значение, а не другие строки метаданных с тем же метаключом:

- `$post_id` — обязательное целое число — идентификатор записи;
- `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите удалить;
- `$meta_value` — необязательное значение смешанного типа — текущее значение метаданных. Если у вас есть несколько записей с одним и тем же метаключом, вы можете указать, какую из них удалить, сопоставив данное метазначение. По умолчанию значение пусто, что приведет к удалению всех метастрок, где найдено соответствие, включая `post_id` и `meta_key`.

В листинге 2.5 мы получаем последний пост и добавляем, обновляем и удаляем различные метаданные этого поста.

Листинг 2.5. Работа с метаданными поста

```
<?php
// получить публикации - вернуть последнюю публикацию
$posts = get_posts(array('numberposts' => '1', 'orderby' =>
    'post_date', 'order' => 'DESC'));
foreach ($posts as $post) {
    $post_id = $post->ID;

    // обновить публичные метаданные поста
    $content = 'You SHOULD see this custom field when editing your latest
post.';
    update_post_meta($post_id, 'bwawwp_displayed_field', $content);

    // обновить скрытые метаданные поста
    $content = str_replace('SHOULD', 'SHOULD NOT', $content);
    update_post_meta($post_id, '_bwawwp_hidden_field', $content);

    // массив имен пользователей-студентов
    $students[] = 'dalya';
    $students[] = 'ashleigh';
    $students[] = 'lola';
    $students[] = 'isaac';
    $students[] = 'marin';
    $students[] = 'brian';
    $students[] = 'nina';
    $students[] = 'cam';

    // добавляем метаданные посту - один ключ с массивом в качестве
    // значения, массив будет сериализован автоматически
    add_post_meta($post_id, 'bwawwp_students', $students, true);
```

```

// Пройтись в цикле по всем студентам и добавить метазапись
// поста для каждого студента
foreach ($students as $student) {
    add_post_meta($post_id, 'bwawwp_student', $student);
}

// получить метаданные поста - получить все метаключи
$all_meta = get_post_meta($post_id);
echo '<pre>';
print_r($all_meta);
echo '</pre>';

// получить метаданные поста - получить 1-й экземпляр ключа
$student = get_post_meta($post_id, 'bwawwp_student', true);
echo 'oldest student: ' . $student;

// удаляем метаданные поста
delete_post_meta($post_id, 'bwawwp_student');
}

```

/*
 Результат выполнения этого листинга должен выглядеть примерно так:

```

Array
(
    [_bwawwp_hidden_field] => Array
        (
            [0] => You SHOULD NOT see this custom field when editing your latest
post.
        )

    [bwawwp_displayed_field] => Array
        (
            [0] => You SHOULD see this custom field when editing your latest post.
        )

    [bwawwp_students] => Array
        (
            [0] => a:7:{i:0;s:5:"dalya";i:1;s:8:"ashleigh";i:2;s:4:"lola";i:3;s:5:
"isaac";i:4;s:5:"marin";i:5;s:5:"brian";i:6;s:4:"nina";i:6;s:5:"cam";}
        )

    [bwawwp_student] => Array
        (
            [0] => dalya
            [1] => ashleigh
            [2] => lola
            [3] => isaac
        )

```

```
[4] => marin
[5] => brian
[6] => nina
[7] => cam
)
)
oldest student: dalya
*/
?>
```

Таблица `wp_comments`

Комментарии могут быть оставлены для любого поста. В таблице `wp_comments` хранятся отдельные и связанные комментарии к любому сообщению. В табл. 2.6 показана структура базы данных для таблицы `wp_comments`.

Таблица 2.6. Структура базы данных для таблицы `wp_comments`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
<code>comment_ID</code>	<code>bigint(20)</code>		No	None	AUTO_INCREMENT
<code>comment_post_ID</code>	<code>bigint(20)</code>		No	0	
<code>comment_author</code>	<code>tinytext</code>	<code>utf8_general_ci</code>	No		
<code>comment_author_email</code>	<code>varchar(100)</code>	<code>utf8_general_ci</code>	No		
<code>comment_author_url</code>	<code>varchar(200)</code>	<code>utf8_general_ci</code>	No		
<code>comment_author_IP</code>	<code>varchar(100)</code>	<code>utf8_general_ci</code>	No		
<code>comment_date</code>	<code>datetime</code>		No	0000-00-00 00:00:00	
<code>comment_date_gmt</code>	<code>datetime</code>		No	0000-00-00 00:00:00	
<code>comment_content</code>	<code>text</code>	<code>utf8_general_ci</code>	No	None	
<code>comment_karma</code>	<code>int(11)</code>		No	0	
<code>comment_approved</code>	<code>varchar(20)</code>	<code>utf8_general_ci</code>	No	1	
<code>comment_agent</code>	<code>varchar(20)</code>	<code>utf8_general_ci</code>	No		
<code>comment_type</code>	<code>varchar(20)</code>	<code>utf8_general_ci</code>	No		
<code>comment_parent</code>	<code>bigint(20)</code>		No	0	
<code>user_id</code>	<code>bigint(20)</code>		No	0	

Функции в каталоге /wp-includes/comment.php

Следующие функции находятся в файле /wp-includes/comment.php.

- ◆ `get_comment` (`$ comment`, `$ output = OBJECT`) — эта функция возвращает данные комментария по идентификатору комментария или объекта комментария. Если комментарий пуст, будет использоваться глобальная переменная комментария, если она установлена:
 - `$comment` — необязательное целое число, строка или объект идентификатора комментария или объекта;
 - `$output` — необязательная строка, определяющая формат вывода. Возможные значения: `OBJECT`, `ARRAY_A` и `ARRAY_N`.
- ◆ `get_comments` (`$args = ''`) — эта функция извлекает список комментариев для определенных постов или отдельного поста. Она вызывает класс `WP_Comment_Query`, который мы рассмотрим в следующей главе. `$args` — необязательный массив или строка аргументов для запроса комментариев. Аргументы по умолчанию:
 - `author_email` — строка адреса электронной почты автора комментария;
 - `ID` — целое число — идентификатор комментария;
 - `karma` — целое число кармы комментария, которое может быть использовано плагинами для проставления оценки;
 - `number` — целое число — количество возвращаемых комментариев. По умолчанию все комментарии;
 - `offset` — целое число комментариев, которые нужно пропустить. По умолчанию 0;
 - `orderby` — строка поля, по которой нужно упорядочить комментарий. Допустимые значения: `comment_agent`, `comment_approved`, `comment_author`, `comment_author_email`, `comment_author_IP`, `comment_author_url`, `comment_content`, `comment_date`, `comment_date_gmt`, `comment_ID`, `comment_karma`, `comment_parent`, `comment_post_ID`, `comment_type` и `user_id`;
 - `order` — строка, задающая вид упорядочивания по выбранному аргументу `orderby`. По умолчанию — `DESC` (по убыванию), также возможно `ASC` (по возрастанию);
 - `parent` — целое число — идентификатор родительского комментария;
 - `post_id` — целое число — идентификатор сообщения, к которому прикреплен комментарий;
 - `post_author` — целое число — идентификатор автора поста, к которому прикреплен комментарий;
 - `post_name` — строка названия поста, к которому прикреплен комментарий;
 - `post_parent` — целое число — идентификатор родительского комментария, к которому прикреплен данный;

- `post_status` — строка статуса поста, к которому прикреплен комментарий;
 - `post_type` — строка типа поста, к которому прикреплен комментарий;
 - `status` — строка статуса комментария. Необязательные значения: `hold` (удержан), `approve` (утвержден), `spam` (спам) или `trash` (удален);
 - `type` — строка типа комментария. Необязательные значения: `'`, `pingback` или `track back`;
 - `user_id` — целое число — идентификатор пользователя, написавшего комментарий;
 - `search` — строка поисковых терминов, по которым вы можете искать комментарии. Выполняет поиск в следующих полях: `comment_author`, `comment_author_email`, `comment_author_url`, `comment_author_IP` и `comment_content`;
 - `count` — логическое значение, которое заставит запрос возвращать количество результатов или сами результаты. Значение по умолчанию — `false`;
 - `meta_key` — метаключ комментария для поиска;
 - `meta_value` — метазначение метаданных комментария для поиска; `meta_key` обязателен.
- ◆ `wp_insert_comment($commentdata)` — эта функция добавляет комментарий в базу данных:
- `$commentdata` — обязательный массив полей комментариев и значений для вставки. Можно вставить следующие поля: `comment_post_ID`, `comment_author`, `comment_author_email`, `comment_author_url`, `comment_author_IP`, `comment_date`, `comment_date_gmt`, `comment_content`, `comment_karma`, `comment_approved`, `comment_agent`, `comment_type`, `comment_parent` и `user_id`.
- ◆ `wp_update_comment($commentarr)` — эта функция обновляет данные комментариев и фильтры, чтобы убедиться, что все обязательные поля заполнены верно перед обновлением в базу данных:
- `$commentarr` — необязательный массив аргументов, содержащий поля комментариев и значения для обновления. Это те же самые аргументы поля, которые только что были перечислены для функции `wp_insert_comment()`.
- ◆ `wp_delete_comment($comment_id, $force_delete = false)` — эта функция удаляет комментарий. По умолчанию она поместит комментарий в корзину, если вы не укажете удалить его навсегда:
- `$comment_id` — обязательное целое число — идентификатор комментария, который вы хотите поместить в корзину или удалить;
 - `$force_delete` — необязательный параметр логического типа, который при значении `true` навсегда удалит комментарий.

В листинге 2.6 продемонстрированы некоторые основные функции для взаимодействия с таблицей `wp_comments`, в частности управление данными комментариев, прикрепленных к посту.

```

<?php
// создать пост в БД
$args = array(
    'post_title'    => 'What should I do tonight?',
    'post_content' => 'Think of something cool to do and make a comment about it!',
    'post_status'  => 'publish'
);
$post_id = wp_insert_post($args);
echo 'post ID: ' . $post_id . ' - ' . $args['post_title'] . '<br>';

// создаем массив комментариев
$comments[] = 'ICE CREAM!!!!';
$comments[] = 'Taco Bell';
$comments[] = 'Get a good night sleep';

// Проход по циклу комментариев
foreach ($comments as $key => $comment) {
    // Вставить комментарии
    $commentdata = array(
        'comment_post_ID' => $post_id,
        'comment_content' => $comments[$key],
    );
    $comment_ids[] = wp_insert_comment($commentdata);
}
echo 'comments:<pre>';
print_r($comments);
echo '</pre>';

// обновить комментарий
$commentarr['comment_ID'] = $comment_ids[0];
$commentarr['comment_content'] = 'Read this entire book';
wp_update_comment($commentarr);

// вставляем комментарий - подкомментарий от родительского идентификатора
$commentdata = array(
    'comment_post_ID' => $post_id,
    'comment_parent' => $comment_ids[0],
    'comment_content' => 'That is a pretty good idea...',
);
wp_insert_comment($commentdata);

// получить комментарии из БД - поиск Taco Bell
$comments = get_comments('search=Taco Bell&number=1');
foreach ($comments as $comment) {

```

```

// вставляем комментарий - подкомментарий комментария
// о Taco Bell (по идентификатору)
$commentdata = array(
    'comment_post_ID' => $post_id,
    'comment_parent' => $comment->comment_ID,
    'comment_content' => '',
);
wp_insert_comment($commentdata);
}

// получить комментарий - количество комментариев к этой записи
$comment_count = get_comments('post_id= ' . $post_id . '&count=true');
echo 'comment count: ' . $comment_count . '<br>';

// получить комментарии - получить все комментарии к этой записи
$comments = get_comments('post_id=' . $post_id);
foreach ($comments as $comment) {
    // обновляем первый комментарий
    if ($comment_ids[0] == $comment->comment_ID) {
        $commentarr = array(
            'comment_ID' => $comment->comment_ID,
            'comment_content' => $comment->comment_content . ' & build some apps!',
        );
        wp_update_comment($commentarr);
        // удаляем все остальные комментарии
    } else {
        // удаляем комментарий
        wp_delete_comment($comment->comment_ID, true);
    }
}

// получить комментарий - количество новых комментариев
$comment_count = get_comments('post_id= ' . $post_id . '&count=true');
echo 'new comment count: ' . $comment_count . '<br>';

// получить комментарий - получить лучший комментарий
$comment = get_comment($comment_ids[0]);
echo 'best comment: ' . $comment->comment_content;

/*
Результат выполнения этого листинга должен выглядеть примерно так:
post ID: 91011 - What should I do tonight?
comments:
Array
(
    [0] => ICE CREAM!!!!
    [1] => Taco Bell

```

```
[2] => Get a good night sleep
)
comment count: 5
new comment count: 1
best comment: Read this entire book & build some apps!
*/
?>
```

Таблица `wp_commentsmeta`

Как и в таблицах `wp_usermeta` и `wp_postmeta`, в этой таблице хранятся произвольные пользовательские данные, связанные с комментарием по полю `comment_id`. В табл. 2.7 показана структура базы данных для таблицы `wp_commentsmeta`.

Таблица 2.7. Структура базы данных для таблицы `wp_commentsmeta`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
meta_id	bigint(20)		No	None	AUTO_INCREMENT
comment_id	bigint(20)		No	0	
meta_key	varchar(255)	utf8_general_ci	Yes	NULL	
meta_value	longtext	utf8_general_ci	Yes	NULL	

Функции из каталога `/wp-includes/comment.php`

Следующие функции находятся в файле `/wp-includes/comment.php`.

- ◆ `get_comment_meta($comment_id, $key = '', $single = false)` — эта функция возвращает метаданные комментария для данного идентификатора комментария:
 - `$comment_id` — обязательное целое число — идентификатор комментария, для которого вы хотите получить метаданные;
 - `$key` — необязательная строка имени метаключа, по которому вы хотите получить метаданные. По умолчанию возвращаются метаданные для всех метаключей для определенного поста;
 - `$single` — значение логического типа, возвращать ли единственное значение или нет. По умолчанию — `false`, т. е. значение будет возвращено в виде массива.
- ◆ `add_comment_meta($comment_id, $meta_key, $meta_value, $unique = false)` — эта функция добавляет метаданные для данного идентификатора комментария:
 - `$comment_id` — обязательное целое число — идентификатор комментария;

- `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите сохранить;
 - `$meta_value` — обязательное значение смешанного типа — целое число, строка, массив или объект;
 - `$unique` — необязательный параметр логического типа, который при значении `true` гарантирует, что метаключ может быть добавлен только один раз для данного идентификатора.
- ◆ `update_comment_meta($comment_id, $meta_key, $meta_value, $prev_value = '')` — эта функция возвращает метаданные комментария для данного идентификатора комментария:
- `$comment_id` — обязательное целое число — идентификатор комментария;
 - `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите сохранить. Если этот метаключ уже существует, он обновит метазначение и присвоит ему текущую строку; если нет, он вставит новую запись;
 - `$meta_value` — обязательное значение смешанного типа — целое число, строка, массив или объект. Массивы и объекты будут автоматически сериализованы;
 - `$prev_value` — необязательное значение смешанного типа — текущее значение метаданных. Если совпадение найдено, оно заменит предыдущее/текущее значение на указанное вами новое значение. Если оставить пустым, новое метазначение заменит первый экземпляр соответствующего ключа. Если у вас есть пять строк метаданных с одним и тем же ключом, и вы не указываете, какую строку обновлять с помощью этого значения, то будет обновлена первая строка и удалены остальные четыре.
- ◆ `delete_comment_meta($comment_id, $meta_key, $meta_value = '')` — эта функция удаляет метаданные комментария для предоставленного идентификатора и соответствующего ключа. Вы также можете указать соответствующее метазначение, если хотите удалить только это значение, а не другие строки метаданных с тем же метаключом:
- `$comment_id` — обязательное целое число — идентификатор комментария;
 - `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите удалить;
 - `$meta_value` — необязательное значение смешанного типа — текущее значение метаданных. Если у вас есть несколько записей с одним и тем же метаключом, вы можете указать, какую из них удалить, сопоставив данное метазначение. По умолчанию значение пусто, что приведет к удалению всех метастрок, где найдено соответствие, включая `post_id` и `meta_key`.

В листинге 2.7 продемонстрированы некоторые основные функции для взаимодействия с таблицей `wp_commentsmeta`.

```

<?php
// получить комментарии - идентификатор последнего комментария
$comments = get_comments('number=1');
foreach ($comments as $comment) {
    $comment_id = $comment->comment_ID;

    // добавляем метаданные комментария для просмотра даты и IP-адреса
    $viewed = array(date("m.d.y"), $_SERVER["REMOTE_ADDR"]);
    $comment_meta_id = add_comment_meta($comment_id, 'bwawwp_view_date',
    $viewed, true);
    echo 'comment meta id: ' . $comment_meta_id;

    // обновить метаданные комментария - изменить формат даты на формат
    // October 23, 2020, 12:00 am вместо 10.23.20
    $viewed = array(date("F j, Y, g:i a"), $_SERVER["REMOTE_ADDR"]);
    update_comment_meta($comment_id, 'bwawwp_view_date', $viewed);

    // получить метаданные комментария - все ключи
    $comment_meta = get_comment_meta($comment_id);
    echo '<pre>';
    print_r($comment_meta);
    echo '</pre>';

    // удалить комментарий
    delete_comment_meta($comment_id, 'bwawwp_view_date');
}

/*
Результат выполнения листинга должен выглядеть примерно так:
comment meta id: 16
Array
(
    [bwawwp_view_date] => Array
        (
            [0] => a:2:{i:0;s:24:"August 11, 2018, 4:16 pm";i:1;s:9:"127.0.0.1";}
        )
)
*/
?>

```

Таблица `wp_terms`

В таблице `wp_terms` хранятся все имена категорий или терминов, которое вы создаете. Каждая запись связана с ее таксономией в таблице `wp_term_taxonomy` посредством `term_id`. Итак, вы знакомы с категориями постов и тегами? Каждая категория или

тег хранятся в этой таблице, и технически они являются таксономиями. Каждый термин, который хранится в столбце `name` (имя), является термином таксономии. Мы расскажем о таксономиях более подробно в *главе 5*, поэтому, если вы не до конца понимаете, что такое таксономия, скоро все будет ясно. В табл. 2.8 показана структура базы данных для таблицы `wp_terms`.

Таблица 2.8. Структура базы данных для таблицы `wp_terms`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
<code>term_id</code>	<code>bigint(20)</code>		No	None	AUTO_INCREMENT
<code>name</code>	<code>varchar(200)</code>		No		
<code>slug</code>	<code>varchar(200)</code>	<code>utf8_general_ci</code>	No		
<code>term_group</code>	<code>bigint(10)</code>		No	0	

Функции в каталоге `/wp-includes/taxonomy.php`

Следующие функции находятся в файле `/wp-includes/taxonomy.php`.

◆ `get_terms($taxonomies, $args = '')` — эта функция получает термины определенной таксономии или массива таксономий:

- `$taxonomies` — обязательная строка или массив таксономии или список таксономий;
- `$args` — необязательная строка или массив аргументов.

Доступные аргументы:

- `orderby` — по умолчанию `name` (по имени). Может быть `name`, `count`, `term_group`, `slug`. Если не указан, то будет использоваться `term_id`. Передача пользовательского значения, отличного от этих, приведет к тому, что термины будут упорядочены по этому пользовательскому значению;
- `order` — `ASC` или `DESC`. Значение по умолчанию `ASC` (по возрастанию);
- `hide_empty` — значением по умолчанию является `true`, которое будет возвращать только термины, прикрепленные к посту. Если установлено значение `false`, вы можете вернуть все термины независимо от того, используются ли они в посте или нет;
- `exclude` — массив или разделенная запятыми или пробелами строка идентификаторов терминов для исключения из результатов запроса. Если указано `include`, то `exclude` будет игнорироваться;
- `exclude_tree` — массив или разделенная запятыми или пробелами строка идентификаторов терминов для исключения из результатов запроса, включая

все дочерние термины. Если задано `include`, то `exclude_tree` будет игнорироваться;

- `include` — массив или разделенная запятыми или пробелами строка идентификаторов терминов, которые надо включить в результаты запроса;
- `number` — количество терминов, которые должен вернуть запрос. Значение по умолчанию — все термины;
- `offset` — число, на которое нужно сместить термины в результате запроса (пропустить);
- `fields` — вы можете указать, хотите ли вы вернуть идентификаторы или имена терминов. По умолчанию установлено значение `all`, которое возвращает массив объектов терминов;
- `slug` — строка, которая будет возвращать любые термины, которые имеют совпадающий фрагмент URL;
- `hierarchical` — включает все дочерние термины, если они прикреплены к сообщениям. По умолчанию установлено значение `true`, поэтому, чтобы не возвращать термины в иерархическом порядке, установите для этого значения в `false`;
- `search` — строка, которая будет возвращать любые термины, имена которых соответствуют указанному значению. Поиск не зависит от регистра;
- `name_like` — строка, которая будет возвращать любые термины, имена которых начинаются с предоставленного значения. Как и при поиске, эта строка нечувствительна к регистру;
- `pad_counts` — если установлено значение `true`, результаты запроса будут включать количество дочерних элементов каждого термина;
- `get` — если установлено значение `all`, возвращает термины независимо от происхождения или от того, являются ли они пустыми;
- `child_of` — если задано значение идентификатора термина, то результаты запроса будут содержать всех потомков предоставленного идентификатора термина. По умолчанию имеет значение `0`, что возвращает все термины;
- `parent` — если задан идентификатор термина, то результаты запроса будут содержать прямые дочерние элементы предоставленного идентификатора термина. По умолчанию используется пустая строка;
- `cache_domain` — позволяет создать уникальный ключ кэша, когда этот запрос хранится в кэше объектов.

◆ `get_term($term, $taxonomy, $output = OBJECT, $filter = 'raw')` — эта функция получает все данные для любого заданного термина:

- `$term` — обязательное целое число или объект термина, который необходимо вернуть;

- `$taxonomy` — обязательная строка таксономии термина, который необходимо вернуть;
 - `$output` — необязательная строка формата вывода. Значением по умолчанию является `OBJECT`, а другими значениями могут быть `ARRAY_A` (ассоциативный массив) или `ARRAY_N` (числовой массив);
 - `$filter` — необязательная строка, сообщающая о том, как контекст должен быть очищен при выводе. Значение по умолчанию — `raw`.
- ◆ `wp_insert_term($term, $taxonomy, $args = array())` — эта функция добавляет новый термин в базу данных:
- `$term` — обязательная строка – термин для добавления или обновления;
 - `$taxonomy` — обязательная строка таксономии, к которой будет добавлен термин;
 - `$args` — необязательный массив или строка аргументов термина для вставки/обновления.

Доступны следующие аргументы:

- `alias_of` — необязательная строка части URL, псевдонимом которого будет термин;
 - `description` — необязательная строка, которая описывает термин;
 - `parent` — необязательное целое число — идентификатор родительского термина, для которого этот термин будет дочерним;
 - `slug` — необязательная строка части URL термина.
- ◆ `wp_update_term($term_id, $taxonomy, $args = array())` — эта функция обновляет существующий термин в базе данных:
- `$term_id` — обязательное целое число — идентификатор термина, который вы хотите обновить;
 - `$taxonomy` — обязательная строка таксономии, с которой связан термин;
 - `$args` — необязательный массив или строка аргументов термина для обновления. Это те же аргументы, что и в `wp_insert_term()`.
- ◆ `wp_delete_term($term, $taxonomy, $args = array())` — эта функция удаляет термин из базы данных. Если термин является родителем других терминов, то дочерние элементы будут обновлены и получат в роли родительского термина родительский термин удаленного термина:
- `$term` — обязательное целое число — идентификатор термина, который вы хотите удалить;
 - `$taxonomy` — обязательная строка таксономии, с которой связан термин;
 - `$args` — необязательный массив для перезаписи значений полей терминов.

Таблица `wp_termmeta`

Начиная с WordPress 4.4, метаданные могут храниться и для терминов. В плагине простого упорядочения таксономии от YIKES, Inc. (oreil.ly/thTgK) метаданные терминов позволяют вам изменить отображение в списках и виджетов ваших категорий и других таксономий.

Структура базы данных для таблицы `wp_termmeta` приведена в табл. 2.9.

Таблица 2.9. Структура базы данных для таблицы `wp_termmeta`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
<code>meta_id</code>	<code>bigint(20)</code>		No	None	AUTO_INCREMENT
<code>term_id</code>	<code>bigint(20)</code>		No	0	
<code>meta_key</code>	<code>varchar(255)</code>	<code>utf8_general_ci</code>	Yes	NULL	
<code>meta_value</code>	<code>longtext</code>	<code>utf8_general_ci</code>	Yes	NULL	

Следующие функции работают аналогично вариантам для метаданных пользователей, постов и комментариев.

◆ `get_term_meta($term_id, $key = '', $single = false)` — эта функция получает метазначение термина для указанного ключа:

- `$term_id` — обязательное целое число — идентификатор термина;
- `$key` — необязательная строка метаключа, значение которого вы хотите вернуть. Если пусто, будут возвращены все метаданные для данного термина;
- `$single` — значение логического типа определяющее, возвращать ли единственное значение или нет. По умолчанию — `false`, т. е. значения будут возвращены в виде массива.

Для одного и того же идентификатора термина может быть несколько метаключей с разными значениями. Если вы установите `$single` в `true`, то получите значение первого ключа; если вы установите `false`, то получите массив значений каждой записи с тем же ключом.

◆ `update_term_meta($term_id, $meta_key, $meta_value, $prev_value = '')` — эта функция обновит метаданные термина и при этом создаст метаданные, если переданный ключ еще не существует:

- `$term_id` — обязательное целое число — идентификатор термина;
- `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите сохранить. Если этот метаключ уже существует, то он обновит метазначение и присвоит ему текущую строку; если нет, он вставит новую запись;

- `$meta_value` — обязательное значение смешанного типа — целое число, строка, массив или объект. Массивы и объекты будут автоматически сериализованы;
- `$prev_value` — необязательное значение смешанного типа — текущее значение метаданных. Если совпадение найдено, оно заменит предыдущее/текущее значение на указанное вами новое значение. Если оставить пустым, новое метазначение заменит первый экземпляр соответствующего ключа. Если у вас есть пять строк метаданных с одним и тем же ключом, и вы не указываете, какую строку обновлять с помощью этого значения, будет обновлена первая строка и удалены остальные четыре.



Эта функция опирается на функцию `update_metadata()`, расположенную в `/wp-includes/meta.php`. Обязательно посмотрите!

- ◆ `add_term_meta($term_id, $meta_key, $meta_value, $unique = false)` — эта функция вставляет новую метазапись в таблицу `wp_termmeta`. Опять же, функция `update_term_meta()` предпочтительна для вставки новых строк, а также их обновления. Если вы хотите убедиться, что данный метаключ используется только один раз для каждого термина, вам следует вызвать эту функцию и установить для параметра `$unique` значение `true`:
 - `$term_id` — обязательное целое число — идентификатор термина;
 - `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите сохранить;
 - `$meta_value` — обязательное значение смешанного типа — целое число, строка, массив или объект;
 - `$unique` — необязательный параметр логического типа, который при значении `true` гарантирует, что метаключ может быть добавлен только один раз для данного идентификатора.
- ◆ `delete_term_meta($term_id, $meta_key, $meta_value = '')` — эта функция удаляет метаданные термина для предоставленного идентификатора термина и соответствующего ключа. Вы также можете указать соответствующее метазначение, если хотите удалить только это значение, а не другие строки метаданных с тем же метаключом:
 - `$term_id` — обязательное целое число — идентификатор термина;
 - `$meta_key` — обязательная строка имени метаключа для метазначения, которое вы хотите удалить;
 - `$meta_value` — необязательное значение смешанного типа — текущее значение метаданных. Если у вас есть несколько записей с одним и тем же метаключом, то вы можете указать, какую из них удалить, сопоставив метазначение. По умолчанию значение пусто, что приведет к удалению всех метастрок, где найдено соответствие, включая поля `term_id` и `meta_key`.

Таблица `wp_term_taxonomy`

В таблице `wp_term_taxonomy` хранится каждый используемый вами тип таксономии. WordPress имеет два встроенных типа таксономии, `category` и `post_tag`, но вы также можете зарегистрировать свои собственные таксономии. Когда новый термин добавляется в таблицу `wp_terms`, он ассоциируется с его таксономией в этой таблице, а также с идентификатором термина таксономии, описанием, родителем и счетчиком. В табл. 2.10 показана структура базы данных для таблицы `wp_term_taxonomy`.

Таблица 2.10. Структура базы данных для таблицы `wp_term_taxonomy`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
<code>term_taxonomy_id</code>	<code>bigint(20)</code>		No	None	AUTO_INCREMENT
<code>term_id</code>	<code>bigint(20)</code>		No	0	
<code>taxonomy</code>	<code>varchar(32)</code>	<code>utf8_general_ci</code>	No		
<code>description</code>	<code>longtext</code>	<code>utf8_general_ci</code>	No	None	
<code>parent</code>	<code>bigint(20)</code>		No	0	
<code>count</code>	<code>bigint(20)</code>		No	0	

Функции в каталоге `/wp-includes/taxonomy.php`

Вы можете найти следующие функции в файле `/wp-includes/taxonomy.php`.

- ◆ `get_taxonomies($args = array(), $output = 'names', $operator = 'and')` — эта функция возвращает список зарегистрированных объектов таксономии или список имен таксономии:
 - `$args` — необязательный массив аргументов для запроса возвращаемых объектов таксономии. Их много, и мы рассмотрим все из них в главе 5;
 - `$output` — необязательная строка `names` (имена) или `objects` (объекты). По умолчанию — `names`, которые возвращают список имен таксономии;
 - `$operator` — необязательная строка со значением либо `and` (и), либо `or` (или). По умолчанию — `and`, означает, что все переданные аргументы должны совпадать. Если установлено значение `or`, любые передаваемые аргументы могут совпадать.
- ◆ `get_taxonomy($taxonomy)` — эта функция сначала проверит, что данная строка параметра является объектом таксономии; если это так, она вернет его:
 - `$taxonomy` — обязательная строка имени объекта таксономии для возврата.
- ◆ `register_taxonomy($taxonomy, $object_type, $args = array())` — эта функция создает или обновляет объект таксономии. Регистрация пользовательских таксо-

номий позволяет действительно расширить WordPress, потому что вы можете классифицировать свои сообщения любым способом, который вам подходит (мы рассмотрим регистрацию таксономий более подробно в *главе 5*):

- `$taxonomy` — обязательная строка имени объекта таксономии для возврата;
- `$object_type` — обязательный массив или строка типов объектов (типов записей, таких как `post` и `page`), к которым будет привязана эта таксономия;
- `$args` — необязательная строка или массив аргументов. Их много, и мы их рассмотрим все в *главе 5*.

Таблица `wp_term_relationships`

Таблица `wp_term_relationships` связывает термин таксономии с постом. Каждый раз, когда вы назначаете категорию или тег для сообщения, она/он связывается с этим сообщением в данной таблице. В табл. 2.11 показана структура базы данных для таблицы `wp_term_relationships`.

Таблица 2.11. Структура базы данных для таблицы `wp_term_relationships`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
<code>object_id</code>	<code>bigint(20)</code>		No	0	
<code>term_taxonomy_id</code>	<code>bigint(20)</code>		No	0	
<code>term_order</code>	<code>int(11)</code>		No	0	

Рассмотрим соответствующие функции:

- ◆ `get_object_taxonomies($object, $output = 'names')` — эта функция возвращает все таксономии, связанные с типом поста или объектом поста:
 - `$object` — обязательный массив, строка или объект имени (имен) типа (типов) или объекта (объектов) публикации;
 - `$output` — необязательная строка `names` (имена) или `objects` (объекты). По умолчанию значение `names`, которые возвращают список имен таксономии.
- ◆ `wp_get_object_terms($object_ids, $taxonomies, $args = array())` — эта функция возвращает термины, связанные с предоставленным идентификатором или идентификаторами объекта поста и предоставленной таксономией:
 - `$object_ids` — обязательная строка или массив идентификаторов объектов для терминов объекта, которые вы хотите вернуть. Передача идентификатора публикации возвращает термины, связанные с этим идентификатором;
 - `$taxonomies` — обязательная строка или массив имен таксономий, из которых вы хотите вернуть термины. Передача в таксономии `post_tag` возвращает термины таксономии `post_tag`;

- `$args` — необязательный массив или строка аргументов, которые изменяют способ возврата данных.

Вот аргументы, которые вы можете изменить:

- `orderby` — по умолчанию `name`; также принимает `count`, `slug`, `term_group`, `term_order` и `none`;
 - `order` — по умолчанию `ASC` (по возрастанию), а также может быть `DESC` (по убыванию);
 - `fields` — по умолчанию `all`; также может быть `id`, `names`, `slug` и `all_with_object_id`. Этот аргумент определяет, какие значения возвращаются.
- ♦ `wp_set_object_terms($object_id, $terms, $taxonomy, $append = false)` — эта функция добавляет термины таксономии к предоставленному идентификатору объекта и таксономии. Она имеет возможность перезаписать все термины или добавить новые термины к существующим. Если термин, переданный в эту функцию, еще не существует, то он будет создан и затем связан с предоставленным идентификатором объекта и таксономией:
- `$object_id` — обязательное целое число — идентификатор объекта (поста), с которым связаны ваши термины;
 - `$terms` — обязательный массив, целое число или строка терминов, которые вы хотите добавить в объект (публикацию);
 - `$taxonomy` — обязательная строка или массив таксономии или таксономий, с которыми вы хотите связать свой термин;
 - `$append` — необязательный аргумент логического типа (`Boolean`), который по умолчанию имеет значение `false` и заменяет любые существующие термины, связанные с идентификатором объекта, новыми предоставленными вами терминами. Если установлено значение `true`, то ваши новые термины будут добавлены к существующим.



Когда-то рассматривалась будущая версия WordPress, где отсутствовала бы таблица `wp_terms`. Столбцы `name` и `slug` `wp_terms` предполагалось переместить в таблицу `wp_terms_taxonomy` и представление MySQL создавалось бы с именем `wp_terms`, к которому можно обратиться, сохранив обратную совместимость для пользовательских запросов. Это обновление было отложено, но было бы неплохо очистить некоторые таблицы вокруг терминов и таксономий.

Хуки: события и фильтры

Разработчики WordPress не даром едят свой хлеб! Хуки — это здорово, и они делают добавление функциональности в плагины и темы WordPress простым и легким.

Существуют два типа хуков — события и фильтры — и они являются двумя наиболее мощными инструментами в WordPress. Мы тратим много времени, работая с событиями и фильтрами. Понимание этого раздела важно для вашего роста как разработчика WordPress.

События

Если хук событий (технически функция `do_action()`) существует в коде, работающем на WordPress, то вы можете вставить свой код, вызвав функцию `add_action()` и передав имя хука события и пользовательскую функцию вместе с кодом, который вы хотите выполнить:

◆ `do_action($tag, $arg);`

Вот доступные аргументы:

- `$tag` — имя выполняемого хука события;
- `$arg` — один или несколько дополнительных аргументов, которые передаются в функцию, вызываемую из функции `add_action()`, ссылающейся на эту функцию `do_action()`.

Что-то непонятно? Просто продолжайте читать...

Вы можете создать свой собственный хук в теме или плагине, добавив свои собственные функции `do_action()`. Однако гораздо чаще вы будете использовать готовые хуки, уже установленные в ядре WordPress или других плагинах. Например, предположим, что вы хотели проверить, вошел ли пользователь в систему при первой загрузке WordPress, но перед отображением какого-либо вывода в браузере.

Вы можете использовать хук `init`, как показано в следующем примере.

Пример

```
<?php
add_action('init', 'my_user_check');

function my_user_check() {
    if (is_user_logged_in()) {
// сделать что-то, потому что пользователь вошел в систему
    }
}
?>
```

Как это работает? В ядре WordPress есть хук событий `do_action (init)`, и мы вызываем функцию `my_user_check()` из функции `add_action()`. В любой момент времени, когда код выполняется, если он попадает в хук события `init`, тот запускает нашу собственную функцию `my_user_check()`, чтобы сделать все, что мы хотим, прежде чем продолжить.



Посетите справочную страницу WordPress (bit.ly/plugin-api) для получения списка наиболее часто используемых хуков WordPress.

Фильтры

Фильтры являются своего рода хуками событий в том смысле, что вы можете использовать их везде, где они есть в WordPress. Однако вместо того, чтобы вставлять свой собственный код, где есть хук или функция `do_action()`, вы фильтруете возвращаемое значение существующих функций, которые вызывают функцию `apply_filters()` в ядре WordPress, плагинах и/или темах. Другими словами, с помощью фильтров вы можете захватить контент до того, как он будет вставлен в базу данных или до того, как он будет отображен в браузере в виде HTML:

◆ `apply_filters($tag, $value, $var);`

- `$tag` — название хука фильтра;
- `$value` — значение, к которому может быть применен фильтр;
- `$var` — любые дополнительные переменные, такие как строка или массив, которые передаются в функцию фильтра.

Если вы выполните поиск в файлах WordPress по названию `apply_filters()`, то обнаружите, что эта функция вызывается повсеместно, и, как хуки событий, ее также можно добавлять и вызывать из любой темы или плагина. В любом месте кода, работающего на вашем сайте WordPress, где вы видите вызываемую функцию `apply_filters()`, вы можете фильтровать значение, возвращаемое этой функцией. Для примера мы собираемся отфильтровать заголовки всех сообщений, прежде чем они будут отображены в браузере. Мы можем подключиться к любым существующим фильтрам через функцию `add_filter()`.

◆ `add_filter($tag, $function, $priority, $accepted_args);`

- `$tag` — имя хука фильтра, который вы хотите фильтровать. Оно должно соответствовать параметру `$tag` вызова функции `apply_filters()`, для которого вы хотите отфильтровать результаты;
- `$function` — имя пользовательской функции, используемой для фактической фильтрации результатов;
- `$priority` — это число устанавливает приоритет, в котором будет выполняться ваша функция `add_filter()`, по сравнению с другими местами в коде, которые могут ссылаться на тот же тег хука фильтра. По умолчанию, это значение равно 10;
- `$accepted_args` — вы можете установить количество параметров, которые может принять ваша пользовательская функция, которая обрабатывает фильтрацию. По умолчанию установлено значение 1, что является параметром `$value` функции `apply_filters()`.

Хорошо, а как будет выглядеть реальный код всего этого? Начнем с добавления фильтра для изменения заголовка любого сообщения, возвращаемого в браузер. Мы знаем о хуке фильтра для `the_title`, который выглядит так:

```
apply_filters('the_title', $title, $id);
```

Здесь `$title` — это заголовок поста, а `$id` — идентификатор поста.

```
<?php
add_filter('the_title', 'my_filtered_title', 10, 2);
function my_filtered_title($value, $id) {
$value = '[' . $value . ']';
return $value;
}
?>
```

Представленный в примере код программы включает любые заголовки сообщений в скобки. Если бы заголовок вашего сообщения был "Привет, мир", теперь он стал бы "[Привет, мир]". Обратите внимание, что мы не использовали аргумент `$id` в нашей пользовательской функции. Если бы мы хотели, то могли бы добавить скобки только для конкретных определенных идентификаторов постов.



Функция `add_action()` предназначена для использования с хуками `do_action()`, а `add_filter()` — с хуками `apply_filters()`, обе функции работают одинаково и взаимозаменяемы. Для удобства чтения все равно рекомендуется указывать правильную функцию в зависимости от того, намереваетесь ли вы вернуть отфильтрованный результат или просто выполнить какой-то код в определенное время.

Среды разработки и хостинг

Для запуска WordPress нужен веб-сервер. В этом разделе описываются основы настройки локальной и удаленной среды для вашего приложения WordPress.

Работа локально

Прежде чем мы перейдем к основам WordPress, мы предлагаем вам настроить локальную среду разработки для запуска вашей инсталляции WordPress. Вы можете запустить WordPress на своем настольном компьютере или ноутбуке, что сделает разработку более эффективной и быстрой. Вы можете запускать программу локально, а не использовать FTP и не разворачивать код из хранилища на веб-сервере. Вы также можете писать и тестировать программы без подключения к Интернету. Таким образом, когда-либо в будущем вы сможете продуктивно поработать с вашим ноутбуком на приреде.

Существует множество онлайн-ресурсов о том, как запускать WordPress локально, поэтому мы не будем подробно останавливаться на них, но в зависимости от вашей ОС и предпочтений вы, скорее всего, настроите MAMP (Mac, Apache, MySQL, PHP), WAMP (Windows, Apache, MySQL, PHP) или LAMP (Linux, Apache, MySQL, PHP). Также посмотрите XAMPP и AMPPS, которые являются кроссплатформенными стеками.

Есть несколько инструментов для локального развертывания веб-сервера, разработанных специально для WordPress. DesktopServer компании ServerPress (serverpress.com) и Local компании Flywheel (local.getflywheel.com) — это попу-

лярные инструменты, которые оптимизируют всю конфигурацию и параметры настройки для локальной установки WordPress через пользовательский интерфейс.

Такие инструменты, как VirtualBox, Vagrant и Docker, также популярны в сообществе WordPress. Эти инструменты особенно хороши для определенных схем автоматического тестирования и развертывания. WordPress.org содержит хороший документ по установке Varying Vagrant Vagrants (VVV) (oreil.ly/P8s4Z), который популярен среди основных участников разработки.

Выбор веб-хостинга

Если вы еще не знаете, что такое веб-хостинг, вам нужно заняться поиском в Google. В двух словах, веб-хостинг — это место, где находится ваш сайт, в частности, каталог и база данных WordPress.

Мы предполагаем, что большинство из вас уже работают, или работали с веб-хостинговой компанией, или, возможно, вы работаете в компании, которая имеет свои собственные внутренние веб-серверы. Выбор подходящего хоста для вашего веб-приложения является ключом к тому, чтобы обеспечить его быструю, безопасную и масштабируемую работу. Существует множество веб-хостинговых компаний, и почти каждая может разместить WordPress, но не все из них оптимизированы специально для WordPress.

"Управляемая" хостинговая компания WordPress будет лучшим выбором для крупномасштабного веб-приложения на платформе WordPress, так как она сосредоточена на предоставлении оптимизированных сред хостинга WordPress. Для крупных веб-сайтов большая часть работы должна быть направлена на кэширование данных из-за структуры базы данных WordPress с хранением метаданных. Если вы используете WordPress в качестве крупной CMS с множеством постов с большим количеством метаданных публикаций на недостаточно мощном сервере, то он начнет "тормозить" и поглощать ресурсы сервера. Хостинг-компании, ориентированные на оптимизированный WordPress, имеют встроенные системы кэширования, которые помогают улучшить доставку контента.

Вы всегда можете использовать плагины для кэширования, если ваш хостинг не выполняет все операции кэширования за вас (*глава 14* посвящена кэшированию и другим вопросам масштабирования, если вы управляете собственной средой). Некоторые управляемые хосты WordPress предлагают другие продвинутые инструменты, такие как автоматические обновления плагинов и сервисы, такие как интеграция CDN, нацеленная на оптимизацию вашей среды WordPress.

Некоторые примеры управляемых хостингов WordPress включают WP Engine, WordPress VIP и Pagely. Есть много других вариантов хостинга. Мы поддерживаем актуальный список хостингов на нашем сайте (bwwwp.com/hosts/).

Среды разработки, интеграции и доставки

Типичный проект веб-сайта будет иметь три среды программного кода: разработка, интеграция и доставка. В идеале все три среды — это отдельные веб-серверы с отдельными базами данных и отдельными инсталляциями WordPress.

Среда разработки (иногда называемая Dev или DEV) — это место, где вы выполняете свою новую работу по написанию программы и ее обслуживанию. Как правило, это локальная среда, которую вы настроили, на основе информации, приведенной ранее в этой главе.

В среде интегрирования (иногда называемой "Тестирование" или "ТЕСТ") проводится тестирование. Данные на тестовом сайте должны быть как можно ближе к "боевым" данным. Это означает, что вы иногда будете экспортировать данные с "живого" сайта, очищать их для удаления личной информации пользователя и импортировать их на тестовый стенд.

Производственная среда (иногда ее называют Live или PRD) — это настоящий веб-сайт, который пользователи посещают и с которым взаимодействуют.

Если у вас нет всех трех программных сред, то по крайней мере, вам нужна отдельная среда для разработки и производственного стенда. Вы никогда не должны обновлять код на производственном веб-сайте без тщательного его тестирования на тестовом сайте.

В идеале вы должны выполнять всю свою разработку локально, а затем помещать свой код в общий репозиторий и развертывать его на сайте разработки, на котором одновременно могут работать несколько разработчиков. Затем, когда программа будет готова к тщательному тестированию, возможно, вашим начальником или клиентом в среде, которая является точной копией производственного сайта, вы развернете свой код на тестовом сайте. Наконец, когда все проверено на промежуточном сайте, вы развертываете свою программу в Production. Сделайте резервные копии рабочего сайта, прежде чем развертывать какой-либо код на случай, если вам понадобится откатить какие-либо обновления. И, наконец, никогда не думайте, что ваш код будет работать без тестирования.

В некоторых управляемых хостингах WordPress, таких как WP Engine, есть автоматические среды тестирования для каждого сайта WordPress, который вы создали, и это действительно очень полезная функция.



Если вы думали о FTP, когда речь шла о "развертывании кода" ранее, вы должны перейти на уровень выше и использовать репозиторий исходного кода, такой как GitHub.

Расширение WordPress

Теперь вы знаете основы того, как настроен WordPress, как хранятся данные и каковы основные инструменты для манипулирования этими данными. Вы познакомились с хуками событий и фильтров, которые являются основным методом расширения WordPress.

Мы рассмотрим больше различных встроенных функций и методов, используемых для взаимодействия с данными WordPress на протяжении всей книги. Глава 3 посвящена API плагинов WordPress, включая некоторые ключевые функции WordPress, которые делают его расширение простым, мощным и последовательным!

Использование плагинов WordPress

Плагины — потрясающая вещь! Если вы не знали, теперь самое время убедиться в этом. Плагины могут помочь вам развернуть полноценное веб-приложение практически без реального знания программирования. Независимо от того, используете ли вы бесплатный плагин, премиальный плагин или создаете свой собственный, плагины могут расширять WordPress и предоставить вам функциональность, необходимую для вашего приложения.

Как мы упоминали ранее, большое преимущество программного обеспечения с открытым исходным кодом заключается в том, что члены сообщества вкладывают средства в улучшение WordPress и часто создают плагины для достижения желаемой функции. Определение плагина, представленное в кодексе WordPress, — это "программа или несколько функций, написанных на языке сценариев PHP, которые добавляют определенный набор функций или сервисов к веб-блогу WordPress, легко интегрируемый через точки доступа и методы, предоставляемые прикладным программным интерфейсом (API) плагина WordPress". Плагины позволяют превратить ваш сайт во что угодно: от простого блога до сайта интернет-магазина, социальной сети и мобильного приложения для iOS и Android.

В стандартную комплектацию любой новой установки WordPress входят два плагина — Hello Dolly и Akismet. Если вы не знали, то плагин Hello Dolly добавляет случайную строчку из песни "Hello Dolly" (из одноименного мюзикла) в верхнюю часть вашей панели мониторинга при каждой загрузке страницы. Это бесполезно, но позволяет понять, как структурировать ваши собственные плагины. Плагин Akismet интегрируется с Akismet.com для автоматической фильтрации спам-комментариев вашего блога. Если Hello Dolly полезен только с точки зрения обучения, то Akismet совершенно необходим на любом сайте с включенными комментариями. У вас всегда есть возможность деактивировать эти плагины или вообще удалить их, если вы не видите необходимости в них на своем сайте.

Через официальный репозиторий плагинов WordPress (oreil.ly/KcDR9) вы можете получить доступ к более чем 55 тыс. плагинов. Поскольку не все плагины находятся там, для получения дополнительной необходимой вам функциональности выполните поиск в Интернете. Многие создатели плагинов могут размещать свои разработки для скачивания через личные или бизнес-сайты, и берут за них плату. Также доступны премиум-плагины, за использование которых вы должны заплатить. Подобно мобильным приложениям, может быть доступна урезанная бесплатная версия премиум-плагина, а также более продвинутая коммерческая версия. Большинство премиальных плагинов также предлагают лицензии для разработчиков,

что позволяет разработчикам, создающим несколько сайтов, платить один раз за файлы плагинов, которые затем могут быть установлены на нескольких инсталляциях WordPress.



Будьте внимательны при поиске премиальных плагинов в Google. Всегда убеждайтесь, что вы получаете плагин с настоящего сайта автора плагина или его официальных репозиториев. На многих сайтах (также называемых "клубами GPL") появились бесплатные версии "пиратских", "взломанных" или "крякнутых" плагинов премиум-класса по резко сниженным ценам. Большинство из этих плагинов включают вредоносное ПО или другой небезопасный код, который вам явно не нужен.

General Public License, версия 2

Независимо от того, как вы приобретаете или получаете плагин WordPress, все плагины WordPress должны соответствовать лицензии General Public License версии 2 (GPLv2). Она говорит о том, что если исходный код *распространяется* (к нему открыт доступ, он продается онлайн и т. д.), то вы можете делать с кодом что угодно, пока переработанный программный продукт сохраняет лицензию GPLv2. Некоторые темы и плагины могут иметь разделенную лицензию, означающую, что HTML, CSS, JavaScript и изображения распространяются под одной лицензией, а файлы PHP — под другой. Некоторые темы и плагины не упоминают лицензию GPLv2 или категорически отрицают ее применение. В их претензиях есть некая юридическая обоснованность, но авторитетные фигуры в сообществе WordPress.org (а именно Мэтт Малленберг) утверждают, что все темы и плагины должны соответствовать GPL. Наша позиция заключается в том, что, если вы хотите вести бизнес в сообществе WordPress, то должны следовать их правилам.

В целом, плагины — отличный способ добавить расширенную функциональность на ваш сайт без необходимости изменения каких-либо основных файлов WordPress. Если вы ищете конкретную функцию, то нужно сначала убедиться, что плагин для реализации этой функции еще не существует. Если подходящего плагина нет, то у вас есть два варианта: вы можете загрузить и изменить существующий плагин или создать новый с нуля.

Установка плагинов WordPress

Чтобы установить плагин для WordPress, просто зайдите на панель администратора WordPress вашего сайта, также известную как бэкэнд. Нажмите на раздел плагинов, как показано на рис. 3.1. Там вы можете поискать плагин в репозитории WordPress или загрузить его туда, если вы уже скачали нужный плагин из хранилища или другого источника. Когда вы закончите поиск и найдете интересующий плагин, нажмите на его название, чтобы установить. После установки плагина у вас есть возможность активировать его. Если вы не активируете плагин, он остается деактивированным в разделе **Plugins** → **Installed Plugins** вашего сайта. Кроме того, имейте в

виду, что многие плагины необходимо будет настроить после активации, и обычно на панели инструментов вы увидите сообщение, напоминающее вам об этом.

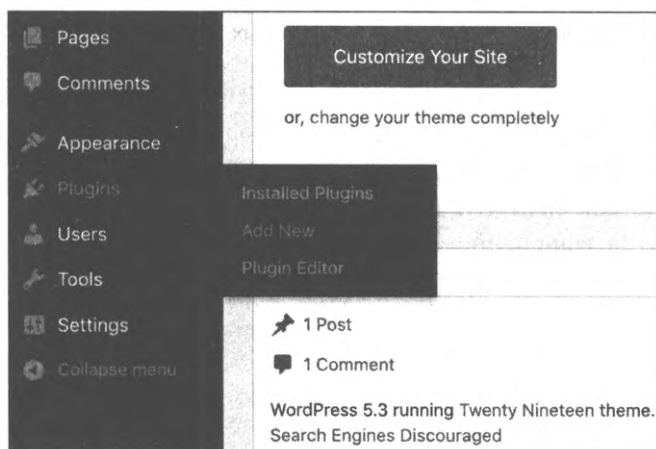


Рис. 3.1. Добавление нового плагина

Если вы загрузили плагин не из официального репозитория плагинов WordPress, то у вас должен быть ZIP-файл, содержащий файлы этого плагина. Чтобы загрузить плагин на свой сайт, в разделе **Plugins** на панели управления нажмите кнопку **Upload**, а затем выберите этот ZIP-файл из того места, где он был сохранен на вашем компьютере. Вам будет предложено активировать плагин, если вы хотите сделать это сразу.

Создание собственного плагина

Истинная мощь WordPress для разработчиков приложений заключается в том, что вы всегда можете создать свой собственный плагин для расширения функциональности WordPress.

Чтобы разработать плагин, сначала создайте новую папку в каталоге `wp-content/plugins` с именем `my-plugin` и пометите в эту папку файл PHP с именем `my-plugin.php`. В `my-plugin.php` вставьте код следующего примера и не стесняйтесь изменять любое из значений.

Пример

```
<?php
/**
 * Plugin Name: My Plugin
 * Plugin URI: https://bwawwp.com/my-plugin/
 * Description: This is my plugin description.
 * Author: messenlehner, strangerstudios
 * Version: 1.0.0
```

```
* Author URI: https://bwwwp.com
* License: GPL-2.0+
* License URI: http://www.gnu.org/licenses/gpl-2.0.txt
*/
?>
```

Сохраните ваш файл `my-plugin.php`. Поздравляем, вы являетесь автором плагина WordPress! Даже если ваш плагин пока еще ничего не делает, вы сможете увидеть его на странице `/wp-admin/plugins.php` и активировать его. Вперед, активируйте его.

Давайте заставим ваш плагин на самом деле делать что-то. Мы добавим текст в нижний колонтитул вашей инсталляции WordPress. Скопируйте и сохраните код следующего примера после информации о плагине.

Пример

```
<?php
function my_plugin_wp_footer() {
    echo 'I read Building Web Apps with WordPress
    and now I am a WordPress Genius!';
}
add_action('wp_footer', 'my_plugin_wp_footer');
?>
```

После того как обновите страницу и перейдете на веб-сайт, вы должны заметить новое сообщение в нижнем колонтитуле. Теперь вы готовы к настоящей игре, и можете настроить этот базовый плагин, чтобы он делал все, что вы хотите. Если вы уже являетесь разработчиком PHP, действуйте! Если вы новичок в PHP и WordPress, то хороший способ дать старт своим навыкам — загрузка и анализ кода других плагинов, чтобы увидеть, как они работают.

Рассмотренный плагин — всего лишь очень простой пример для начала. Далее мы проанализируем больше примеров, которые вы сможете использовать в любом из плагинов, разрабатываемых на протяжении всей книги.

Структура файла плагина приложения

Когда вы создаете веб-приложение с WordPress, мы рекомендуем иметь один основной плагин приложения для хранения базовых функций вашего приложения. Со стороны темы (описанной в *главе 4*) вы будете хранить большую часть кода внешнего интерфейса вашего приложения в активной теме.

Некоторые плагины выполняют только одну или две функции, и один-единственный файл `*.php` — это все, что вам нужно для достижения цели. Ваш основной плагин приложения, вероятно, будет намного сложнее: с файлами ресурсов (CSS, изображения и шаблоны), включенными библиотеками, файлами классов и, возможно, тысячами строк кода, которые вы захотите объединить в несколько файлов.

Далее показана предлагаемая нами структура каталогов и файлов для основного плагина приложения, в качестве примера выбран плагин SchoolPress. Не все эти папки и файлы могут быть необходимы сразу. Мы добавляем их в плагин по мере необходимости:

- ◆ /plugins/schoolpress/adminpages/
- ◆ /plugins/schoolpress/classes/
- ◆ /plugins/schoolpress/css/
- ◆ /plugins/schoolpress/css/admin.css
- ◆ /plugins/schoolpress/css/frontend.css
- ◆ /plugins/schoolpress/js/
- ◆ /plugins/schoolpress/images/
- ◆ /plugins/schoolpress/includes/
- ◆ /plugins/schoolpress/includes/lib/
- ◆ /plugins/schoolpress/includes/functions.php
- ◆ /plugins/schoolpress/pages/
- ◆ /plugins/schoolpress/services/
- ◆ /plugins/schoolpress/scheduled/
- ◆ /plugins/schoolpress/schoolpress.php

Каталог /adminpages/

Поместите файлы *.php для любой страницы панели мониторинга, которую вы добавляете с помощью плагина, в каталоге /adminpages/. Далее приведен пример, как вы можете добавить страницу панели мониторинга и загрузить ее из каталога /adminpages/.

Пример

```
<?php
// добавляем меню SchoolPress с функцией страницы отчетов
function sp_admin_menu() {
    add_menu_page(
        'SchoolPress',
        'SchoolPress',
        'manage_options',
        'sp_reports',
        'sp_reports_page'
    );
}
```

```

add_action('admin_menu', 'sp_admin_menu');
// функция для загрузки страницы администрирования
function sp_reports_page() {
    require_once dirname(__FILE__) . "/adminpages/reports.php";
}
?>

```

Каталог /classes/

Поместите определения классов PHP в каталог /classes/. В общем случае, каждый файл в этом каталоге должен содержать только одно определение класса. Файлы классов должны иметь имена, такие как class.<ClassName>.php, где ClassName — это имя, данное классу.

Каталог /css/

Поместите файлы CSS, используемые вашим плагином, в каталог /css/. Разделите ваш CSS на файлы admin.css и frontend.css в зависимости от того, влияет ли CSS на панель управления WordPress или на что-то в пользовательском интерфейсе (фронтенде). Например, вы также можете поместить библиотеки CSS, необходимые для поддержки включенной библиотеки JavaScript, в эту папку.

Вот пример некоторого кода, чтобы применить стили admin.css и frontend.css из папки CSS плагина.

Пример

```

<?php
function sp_load_admin_styles() {
    wp_enqueue_style(
        'schoolpress-plugin-admin',
        plugins_url('css/admin.css', __FILE__),
        array(),
        SCHOOLPRESS_VERSION,
        'screen'
    );
}

add_action('admin_enqueue_scripts', 'sp_load_admin_styles');

function sp_load_frontend_styles() {
    wp_enqueue_style(
        'schoolpress-plugin-frontend',
        plugins_url('css/frontend.css', __FILE__),
        array(),
        SCHOOLPRESS_VERSION,

```

```

        'screen'
    );
}
add_action('wp_enqueue_scripts', 'sp_load_frontend_styles');
?>

```

Хук `admin_enqueue_scripts` служит для сценариев и стилей, предназначенных для панели администратора, а хук `wp_enqueue_scripts` — для сценариев и стилей фрон-тенда.

Код CSS, который влияет на компоненты панели мониторинга WordPress, должен находиться в файле `admin.css`. Код CSS, который влияет на внешний интерфейс сайта, должен входить в `frontend.css`, но будьте осторожны при добавлении правил CSS в файл `frontend.css`. При добавлении стилей внешнего интерфейса в файлы плагинов сначала спросите себя, должны ли написанные вами правила CSS идти в тему приложения, поскольку бо́льшая часть кода в стиле внешнего интерфейса должна обрабатываться вашей основной темой.

Код CSS, который будет добавлен в CSS-файл плагина, — это, как правило, стили макета, которые подойдут независимо от того, какая тема загружена. Представьте, что на вашем сайте вообще нет темы или CSS. Каким будет минимальный CSS, необходимый для того, чтобы HTML-код, сгенерированный вашим плагином, имел смысл? Вам стоит ожидать, что тема будет построена поверх кода CSS и переопределит его.

Например, файл `frontend.css` вашего плагина никогда не должен включать стили, определяющие цвета. Тем не менее может быть уместен стиль, в котором аватар имеет ширину 64 пиксела и прикреплен к левому краю.

Каталог `/js/`

Поместите в эту папку любые файлы JavaScript, необходимые для вашего плагина. Опять же, вы можете разделить файлы на `admin.js` и `frontend.js` в зависимости от того, где необходим JavaScript.

Вы также можете добавить в эту папку сторонние библиотеки JavaScript, которые вы используете. Как правило, они должны быть добавлены в подпапку каталога `/js/`. Вот пример кода для загрузки файлов `admin.js` и `frontend.js` из каталога `/js/` вашего плагина.

Пример

```

<?php
function sp_load_admin_scripts() {
    wp_enqueue_script(
        'schoolpress-plugin-admin',
        plugins_url('js/admin.js', __FILE__),
        array('jquery'),

```

```

        SCHOOLPRESS_VERSION
    );
}
add_action('admin_enqueue_scripts', 'sp_load_admin_scripts');

function sp_load_frontend_scripts() {
    wp_enqueue_script(
        'schoolpress-plugin-frontend',
        plugins_url('js/frontend.js', __FILE__),
        array('jquery'),
        SCHOOLPRESS_VERSION
    );
}
add_action('wp_enqueue_scripts', 'sp_load_frontend_scripts');
?>

```

Как и при работе с кодом CSS, может быть трудно определить, следует ли включать какой-либо фрагмент кода JavaScript в файл JavaScript плагина или файл JavaScript темы. В общем случае файлы JavaScript, поддерживающие тему (например, эффекты слайдера и эффекты меню), должны идти в тему, а файлы JavaScript, поддерживающие плагин (например, код Ajax), добавляются в плагин. На практике, однако, вы столкнетесь с ситуацией, что ваш плагин использует JavaScript, определенный в вашей теме, и наоборот.



Вы можете использовать одну и ту же функцию обратного вызова для применения как ваших файлов JavaScript, так и таблиц стилей CSS. Вы можете задействовать функции `wp_enqueue_script()` и `wp_enqueue_style()` во время хуков `wp_enqueue_scripts` и `admin_enqueue_scripts`. Хука `wp_enqueue_styles` или `admin_enqueue_styles` не существует.

Каталог `/images/`

Поместите все изображения, необходимые вашему плагину, в каталог `/images/`.

Каталог `/includes/`

Каталог `/includes/` является своего рода ловушкой для файлов `*.php`, которые нужны вашему плагину. Единственный PHP-файл в корневой папке вашего плагина должен быть основным файлом плагина `schoolpress.php`. Все остальные файлы `*.php` должны находиться в одной из других папок. Если ни одна из них не подходит, создайте дополнительную папку и поместите ее в папку `/includes/`.

Это стандартная процедура добавления файла `functions.php`, `common.php` или `helpers.php` для включения любого вспомогательного PHP-кода, используемого вашим плагином. Этот каталог должен содержать любые небольшие скрипты, которые не играют центральной роли в логике или функциональности вашего плагина, но необходимы для его поддержки. Примерами могут быть функции для обрезки текста,

функции для генерации случайных строк или другие подобные фреймворку функции, которые еще не доступны через базовую функцию WordPress.

Каталог `/includes/lib/`

Поместите сторонние библиотеки, необходимые для вашего приложения, в каталог `/includes/lib/`.

Каталог `/pages/`

Поместите любой PHP-код, связанный со страницами внешнего интерфейса, добавленными вашим плагином, в каталог `/pages/`. Страницы внешнего интерфейса, как правило, добавляются с помощью шорткодов (shortcode), которые вы бы встраивали в стандартную страницу WordPress для отображения нужного контента.

В следующем примере показано, как написать шорткод, который можно разместить на странице WordPress для создания страницы из вашего плагина. Ключевое слово `preheader` (предзаголовок) здесь — это фрагмент кода, который выполняется до загрузки функции `wp_head()` и, следовательно, до отправки любых заголовков или кода HTML в браузер. Функция шорткода далее выводит HTML на фактическую страницу в месте шорткода.

Поместите этот код в каталог `/plugins/<your plugin folder>/pages/stub.php` и затем включите его (обычно используя функцию `require_once()`) из основного файла плагина. Затем добавьте шорткод `[sp_stub]` на страницу вашего сайта WordPress.

Пример

```
<?php
// предзаголовок
function sp_stub_preheader() {
    if (!is_admin()) {
        global $post, $current_user;
        if (!empty($post->post_content) && strpos
            ($post->post_content, '[sp_stub]') !== false) {
            /*
             * Введите здесь свой код предзаголовка.
             */
        }
    }
}

add_action('wp', 'sp_stub_preheader', 1);

// шорткод [sp_stub]
function sp_stub_shortcode() {
    ob_start();
    ?>

    Поместите ваш код HTML и т.п. здесь.
```



```

<?php
$temp_content = ob_get_contents();
ob_end_clean();
return $temp_content;
}
add_shortcode('sp_stub', 'sp_stub_shortcode');
?>

```

В коде предварительного заголовка мы сначала проверяем, что страница загружается не администратором, используя конструкцию `!is_admin()`; в противном случае этот код может выполняться, когда мы редактируем сообщение на панели инструментов. Затем мы ищем строку `[sp_stub]` в содержимом глобальной переменной `$post`. Эта функция подключается к хуку `wp`, который запускается после того, как WordPress делает переменную `$post` глобальной для текущей страницы, но до вывода любых заголовков или HTML.

Код предварительного заголовка можно использовать для проверки разрешений, обработки отправленных форм или подготовки любого кода, необходимого для страницы. В модели MVC это будет ваша модель и/или код контроллера. Поскольку этот код запускается до вывода любых заголовков, вы можете безопасно перенаправить пользователей на другую страницу. Например, вы можете с помощью вызова функции `wp_redirect()` перенаправить их на страницу входа или регистрации, если у них нет доступа для просмотра страницы.

В функции шорткода мы используем `ob_start()`, `ob_get_contents()` и `ob_end_clean()`, которые являются функциями PHP для буферизации вывода в переменную. Это означает, что код между предыдущим тегом `?>` и тегом `<?php` помещается в переменную `$temp_content`, а не выводится по мере обработки (что вывело бы содержимое вашего тега `<html>`). Это не обязательно: вы могли бы просто определить функцию `$temp_content()` и с помощью PHP добавлять значения в эту строку. Буферизация вывода позволяет нам сделать код более похожим на шаблон, смешивая HTML и PHP, которые легче читать.

Каталоги `/services/` и `/scheduled/`

Поместите любой PHP-код для вызовов Ajax в каталог `/services/`.

В каталог `/scheduled/` поместите любой PHP-код, который связан с заданиями cron, или код, который должен запускаться через запланированные интервалы.

Файл `schoolpress.php`

Это основной файл плагина. Для небольших плагинов, это может быть единственный необходимый файл. Для больших плагинов основной файл плагинов будет содержать только операторы `include`, определение констант и некоторые комментарии о том, какие другие файлы содержат код, который вы можете использовать.

Дополнения к существующим плагинам

Плагин или фрагмент программы, работающий на WordPress и распространяемый¹, должен иметь открытый исходный код и распространяться по лицензии GPL. Значит, вы можете взять любой плагин в репозитории, изменить имя и выпустить его как совершенно новый плагин. Это может привести к "драке в баре", поэтому мы советуем вам не "размножать" подобные плагины, если вы не планируете улучшать и поддерживать новый плагин.

Допустим, вы нашли плагин, который делает 95% того, что вам нужно, но ему необходимо еще несколько строк кода, чтобы добраться до 100%. В таком случае подумайте о создании *дополнения* для плагина.

Большинство хорошо разработанных плагинов будут иметь свои собственные хуки и фильтры, которые могут позволить другим разработчикам создавать дополнения для плагина. Можно разработать новый плагин для использования хуков и фильтров в WordPress, а можно создать плагин для реализации хуков и фильтров в других плагинах. В некоторых случаях вам может понадобиться вмешаться в код оригинального плагина, чтобы сделать то, что вы хотите, что совершенно нормально, но, возможно, достаточно будет самостоятельно добавить несколько хуков или файлов там, где они вам нужны.

Случаи из практики и примеры

Итак, что мы должны создать с помощью бесплатных и премиальных плагинов, которые только что упомянули? Давайте добавим к сообществу WordPress новое — *SchoolPress*.

Каждый учитель будет администратором своей группы и может легко добавлять в нее учеников. Студенты могут участвовать в групповой деятельности или "Стене класса", как мы ее назовем. С *BuddyPress* учащиеся могут добавлять друг друга в друзья, подписываться на своих друзей или учителей и в личных сообщениях задавать своим учителям вопросы.

С *BadgeOS* и надстройкой *BadgeOS Community* учителя могут создавать забавные поощрительные значки для своих учащихся, за выполнение различных домашних заданий и проектов, а ученики могут поделиться значками с друзьями в других социальных сетях.

И мы можем использовать *Gravity Forms*, чтобы облегчить студентам отправку домашней работы.

¹ В контексте GPL распространение означает продажу вашего исходного кода или предложение его для скачивания на веб-сайте, таком как репозиторий плагинов WordPress.org. Код, который вы лично устанавливаете для кого-то, не должен наследовать лицензию GPL.

Цикл WordPress

Великий и мощный *цикл WordPress* — это то, что заставляет WordPress отображать свои сообщения. В зависимости от того, какой файл шаблона темы вызывается при навигации по веб-сайту, WordPress делает запрос в базу данных и извлекает сообщения, которые необходимо вернуть конечному пользователю, а затем проходит по ним циклом.

Наиболее правильно созданные темы WordPress обычно имеют следующие файлы, которые содержат цикл WordPress:

- ◆ index.php;
- ◆ archive.php;
- ◆ category.php;
- ◆ tag.php;
- ◆ single.php;
- ◆ page.php.

Если вы откроете любой из этих файлов, они будут содержать код, который может выглядеть примерно так, как показано в следующем примере.

Пример

```
<?php
if (have_posts()) {
    while (have_posts()) {
        the_post();
        the_title('<h2>', '</h2>');
        the_content();
    }
} else {
    // показать сообщение в духе "извините, нет сообщений!"
}
?>
```

Функция `have_posts()` проверяет, есть ли сообщения, по которым надо пройти в цикле, и если это так, иницируется цикл `while`. Функция `the_post()`, вызываемая первой в каждой итерации цикла, устанавливает сообщение со всеми его глобальными переменными, чтобы конкретные специфические данные поста могли отображаться конечному пользователю.

Глобальные переменные WordPress

Глобальные переменные — это переменные, которые вы можете определить и затем использовать в любом месте в оставшейся части вашего программного кода.

В WordPress есть несколько встроенных глобальных переменных, которые действительно могут помочь вам сэкономить много времени и ресурсов при написании кода.

Чтобы отобразить полный список всех доступных вам глобальных переменных, выполните следующий код:

```
<?php
echo '<pre>';
print_r($GLOBALS);
echo '</pre>';
?>
```

Чтобы получить доступ к глобальной переменной в любом пользовательском коде, который вы пишете, примените следующую конструкцию:

```
<?php
global $global_variable_name;
?>
```

Некоторые глобальные переменные доступны вам только в зависимости от того, где вы находитесь в WordPress. Вот краткий список некоторых наиболее популярных глобальных переменных:

- ◆ `$post` — объект, который содержит все данные публикации из таблицы `wp_posts` для текущей публикации, в которой вы находитесь в цикле WordPress.
- ◆ `$authordata` — объект со всеми данными автора текущего сообщения, в котором вы находитесь в цикле WordPress.

Класс `$wpdb`

Класс `$wpdb` предназначен для непосредственного взаимодействия с базой данных. После глобализации вы можете использовать `$wpdb` в пользовательских функциях для выбора, обновления, вставки и удаления записей базы данных. Если вы новичок в WordPress и не знакомы со всеми функциями для взаимодействия с базой данных, `$wpdb` станет вашим лучшим другом.

Запросы на основе `$wpdb` также полезны, когда вам нужно управлять пользовательскими таблицами, которые требуются вашему приложению, или выполнять сложный запрос (возможно, объединение нескольких таблиц) быстрее, чем основные функции WordPress будут выполняться самостоятельно. Пожалуйста, не думайте, что встроенные функции WordPress для запросов к базе данных работают медленно. Если вы точно не знаете, что делаете, то захотите применить встроенные функции для получения сообщений, пользователей и метаданных. Ядро WordPress хорошо умеет оптимизировать запросы и кэшировать результаты этих вызовов, что будет неплохо работать на всех используемых вами плагинах. Однако в определенных ситуациях вы можете немного сэкономить время, выполнив собственный запрос. Мы рассмотрим несколько подобных примеров в *главе 14*.

Пользовательские таблицы базы данных

В SchoolPress мы храним отношение выполненных и назначенных заданий учащихся в пользовательской таблице. Это делает основные таблицы WordPress немного чище² и позволяет нам легко реализовать такие запросы, как "выбрать все задачи Исаака".

Чтобы добавить нашу таблицу в базу данных, нам нужно написать SQL-запрос с командой `CREATE TABLE` и направить его к базе данных WordPress. Вы можете использовать либо метод `$wpdb->query()`, либо функцию `dbDelta()` из ядра WordPress.

Нам нужно сделать несколько вещей, чтобы отслеживать наши пользовательские таблицы. Мы хотим сохранить `db_version` для нашего плагина приложения, чтобы мы знали, с какой версией схемы базы данных мы работаем в случае ее обновления между версиями. Мы также можем проверить версию, поэтому запускаем установку SQL только один раз для каждой версии. Еще одна распространенная практика заключается в хранении имени своей таблицы в качестве свойства `$wpdb`, чтобы сделать запрос чуть позже.

В листинге 3.1 приведена часть кода нашей функции настройки базы данных для приложения SchoolPress.

Листинг 3.1. Настройка базы данных для SchoolPress

```
<?php
// настроить базу данных для функции приложения SchoolPress
function sp_setupDB() {
    global $wpdb;

    // шоткаты для таблиц базы данных SchoolPress
    $wpdb->schoolpress_assignment_submissions = $wpdb->prefix.
        'schoolpress_assignment_submissions';

    $db_version = get_option('sp_db_version', 0);

    // создаем таблицы при новой установке
    if (empty($db_version)) {
        global $wpdb;

        $sqlQuery = "

        CREATE TABLE '" . $wpdb->schoolpress_assignment_submissions. "' (
            `assignment_id` bigint(11) unsigned NOT NULL,
            `submission_id` bigint(11) unsigned NOT NULL,
```

² Синхронизированная запись в таблицах `wp_usermeta` и `wp_postmeta` необходима для обеспечения точно такой же возможности поиска для одной таблицы `wp_schoolpress_assignment_submissions`.

```

    UNIQUE KEY `assignment_submission` (`assignment_id`,`submission_id`),
    UNIQUE KEY `submission_assignment` (`submission_id`,`assignment_id`)
  )
  ";

  require_once ABSPATH . 'wp-admin/includes/upgrade.php';
  dbDelta($sqlQuery);

  $db_version = '1.0';
  update_option('sp_db_version', '1.0');
}
}
add_action('init', 'sp_dbSetup', 0);
?>

```

Поскольку функция `sp_dbSetup()` запускается в `init` рано (приоритет 0), ярлыки таблиц доступны для любого другого кода, который вы запускаете. Вы не всегда можете использовать префикс `wp_`, поэтому свойство префикса `$wpdb->` служит для получения префикса базы данных для инсталляции WordPress.

Версия базы данных для приложения SchoolPress хранится в таблице параметров WordPress. Мы получаем значение из опций, и если оно пустое, то запускаем код для создания наших пользовательских таблиц. Оператор SQL `CREATE TABLE` здесь довольно стандартный. Чтобы убедиться, что команды работают, всегда старайтесь запускать их непосредственно в базе данных MySQL до того, как переместите их в код вашего плагина.

С помощью функции `dbDelta()` мы создаем таблицу базы данных. Эта функция создает новую таблицу, если она не существует. Или, если таблица с таким именем уже существует, она вычисляет правильный запрос `ALTER TABLE`, чтобы старая таблица соответствовала новой структуре.

Чтобы стала доступной функция `dbDelta()`, вы должны обязательно включить файл `wp-admin/includes/upgrade.php`, поскольку этот файл загружается только при необходимости. Затем передайте функции `dbDelta()` в качестве аргумента выражение SQL для запроса `CREATE TABLE`. Формат SQL здесь немного строже, чем общий формат MySQL (взят из Кодекса WordPress о создании таблиц с помощью плагинов (bwwawp.com/create-table)):

- ◆ Каждое поле в выражении SQL должно быть на отдельной строке.
- ◆ Между словами `PRIMARY KEY` и определением вашего основного ключа должно быть два пробела.
- ◆ Вместо ключевого слова `INDEX` следует указать ключевое слово `KEY`, в выражении должен быть хотя бы один `KEY`.
- ◆ Имена полей нельзя заключать в апострофы или обратные кавычки.

Выполнение запросов

Создавать таблицы предпочтительнее с помощью функции `dbDelta()`, поскольку она автоматически обновляет более старые версии ваших таблиц, но вы также можете выполнить запрос `CREATE TABLE`, используя конструкцию `$wpdb->query($sqlQuery);`.

Метод `$wpdb->query()` позволяет выполнить любой допустимый оператор SQL. Метод `query()` устанавливает множество свойств для объекта `$wpdb`, которые полезны для отладки или отслеживания ваших запросов:

- ◆ `$wpdb->result` — содержит необработанный результат вашего запроса SQL;
- ◆ `$wpdb->num_queries` — увеличивается при каждом запуске запроса;
- ◆ `$wpdb->last_query` — содержит последний запущенный SQL-запрос;
- ◆ `$wpdb->last_error` — содержит строку с последней сгенерированной ошибкой SQL, если она была;
- ◆ `$wpdb->insert_id` — содержит идентификатор, созданный при последнем успешном запросе `INSERT`;
- ◆ `$wpdb->rows_affected` — равен количеству затронутых строк;
- ◆ `$wpdb->num_rows` — равен количеству строк в результате запроса `SELECT`;
- ◆ `$wpdb->last_result` — содержит массив объектов строк, созданных с помощью функции PHP `mysql_fetch_object()`.

Возвращаемое значение метода `$wpdb->query()` зависит от первых результатов запроса и от того, был ли запрос успешным:

- ◆ Если запрос не выполнен — возвращается *false*. Вы можете проверить это, используя следующий код: `if($wpdb->query($query) === false) { wp_die("it failed!"); }`.
- ◆ По запросам `CREATE`, `ALTER`, `TRUNCATE` и `DROP` — возвращается необработанный результат MySQL.
- ◆ Для запросов `INSERT`, `UPDATE`, `DELETE`, `REPLACE` и `SELECT` — возвращается количество затронутых строк.

Экранирование в запросах к базе данных

Известно, что значения, передаваемые в метод `query()`, не экранируются автоматически. Поэтому вам всегда нужно будет избегать ненадежного ввода при непосредственном вызове метода `query()`.

Существуют два основных способа экранирования значений в ваших SQL-запросах: вы можете обернуть свои переменные в функцию `esc_sql()` (листинг 3.2) или сформировать запрос с помощью метода `$wpdb->prepare()`.

```
global $wpdb;
$user_query = $_REQUEST['uq'];

$sqlQuery = "SELECT user_login FROM $wpdb->users WHERE
user_login LIKE '%" . esc_sql($user_query) . "%' OR
user_email LIKE '%" . esc_sql($user_query) . "%' OR
display_name LIKE '%" . esc_sql($user_query) . "%'
";
$user_logins = $wpdb->get_col($sqlQuery);

if(!empty($user_logins))
{
    echo "<ul>";
    foreach($user_logins as $user_login)
    {
        echo "<li>$user_login</li>";
    }
    echo "</ul>";
}
```

В качестве альтернативы вы можете создать запрос с помощью метода `prepare()`, который работает аналогично функциям `sprintf()` и `printf()` в PHP. Применение метода `prepare()` предпочтительнее, когда это возможно, поскольку он лучше экранирует данные, а также помогает избежать проблем, связанных с неуместными одинарными кавычками.

Метод `prepare()` класса `$wpdb`, расположенный в файле `wp-includes/wp-db.php`, принимает два или более параметров:

- ◆ `$query` — строка вашего пользовательского оператора SQL с заполнителями для каждого динамического значения;
- ◆ `$args` — один или несколько дополнительных параметров, которые будут использоваться для замены заполнителей в вашем операторе SQL.

В строке оператора SQL допускаются следующие директивы:

- ◆ `%d` (целое число);
- ◆ `%f` (число с плавающей запятой);
- ◆ `%s` (строка);
- ◆ `%%` (буквальное назначение процента — аргумент не требуется).

Директивы `%d`, `%f` и `%s` следует оставлять без кавычек, и для каждого заполнителя должен быть передан соответствующий аргумент. Литералы (%) как часть запроса должны быть правильно написаны как `%%`.


```
$sqlQuery = $wpdb->prepare("SELECT user_login FROM $wpdb->users WHERE
user_login LIKE %s OR
user_email LIKE %s OR
display_name LIKE %s", $user_query, $user_query, $user_query);
$user_logins = $wpdb->get_col($sqlQuery);
```



Если вы укажете `$wpdb->prepare()` без включения параметра `$args`, то получите предупреждение PHP: Отсутствует аргумент 2 для `wpdb::prepare()`. Если в вашем SQL нет значения-заполнителя, то вам не нужен `prepare()`.

Символ процента `%` необходим в SQL как подстановочный знак в инструкциях `SELECT` при наличии ключевого слова `LIKE`. Поэтому запрос `user_login LIKE %coleman%` возвращает пользователей с такими логинами, как "jcoleman", "jasoncoleman" и "coleman1982". Чтобы сохранить *литеральный* знак процента на своих местах с помощью метода `prepare()`, нам нужно удвоить его до `%%`, что в конечном запросе будет преобразовано в один `%`.

Еще один знак `%` указан рядом с `%s`. Это заполнитель, на который наш параметр `$user_query` будет заменен после экранирования.

Возможно, вы заметили, что в приведенном примере мы использовали метод `$wpdb->get_col()`. В WordPress существует много полезных методов для объекта `$wpdb` для `SELECT`, `INSERT` и других распространенных запросов MySQL.

Запросы **SELECT** с `$wpdb`

У объекта WordPress `$wpdb` есть несколько полезных методов для выбора массивов, объектов, строк, столбцов или даже отдельных значений из базы данных MySQL с помощью SQL-запросов.

Метод `$wpdb->get_results($query, $output_type)` выполнит ваш запрос и вернет массив `last_results`, включая все строки из вашего SQL-запроса в указанном типе вывода. По умолчанию результатом будет "численно проиндексированный массив объектов строк".

Вот полный список типов вывода из Кодекса WordPress:

- ◆ **OBJECT** — в качестве результата выведет численно проиндексированный массив объектов строк;
- ◆ **OBJECT_K** — результат будет выведен в виде ассоциативного массива объектов строк со значениями из первого столбца в качестве ключей (дубликаты будут отброшены);
- ◆ **ARRAY_A** — результат будет выведен в виде численно проиндексированного массива ассоциативных массивов с именами столбцов в качестве ключей;
- ◆ **ARRAY_N** — результат будет выведен как численно проиндексированный массив численно проиндексированных массивов.

В следующем примере кода показано, как использовать массив, возвращенный методом `$wpdb->get_results()` для типа вывода OBJECT.

Пример

```
<?php
global $wpdb;
$sqlQuery = "SELECT * FROM $wpdb->posts
            WHERE post_type = 'assignment'
            AND post_status = 'publish' LIMIT 10";
$assignments = $wpdb->get_results($sqlQuery);

// строки хранятся в массиве, используйте foreach для их обхода
foreach ($assignments as $assignment) {
    // каждый элемент является объектом с именами свойств, равными именам столбцов
    <h3><?php echo $assignment->post_title;?></h3>
    <?php echo apply_filters("the_content", $assignment->post_content);?>
    <?php
}
?>
```

Метод `$wpdb->get_col($query, $column_offset = 0)` вернет массив значений из первого столбца результатов MySQL. Параметр `column_offset` можно использовать для извлечения других столбцов из результатов (0 — первый, 1 — второй и т. д.).

Эта функция чаще всего применяется для получения идентификаторов из таблицы базы данных для использования в другом вызове функции или запросе к базе данных, как показано в следующем примере.

Пример

```
<?php
global $wpdb;
$sqlQuery = "SELECT ID FROM $wpdb->posts
            WHERE post_type = 'assignment'
            AND post_status = 'publish'
            LIMIT 10";
// получение идентификаторов
$assignment_ids = $wpdb->get_col($sqlQuery);

// результат - массив, пройдем циклом по нему
foreach ($assignment_ids as $assignment_id) {
    // у нас есть идентификатор, мы можем использовать get_post,
    // чтобы получить больше данных
    $assignment = get_post($assignment_id);
}
?>
```

```

<h3><?php echo $assignment->post_title;?></h3>
<?php echo apply_filters("the_content", $assignment->post_content);?>
<?php
}
?>

```

Обратите внимание, что мы помещаем строку `global $wpdb;` в большинстве наших примеров, чтобы еще раз показать, что вам нужно убедиться, что `$wpdb` находится в области видимости, прежде чем вызывать один из его методов. На практике эта строка обычно находится в начале функции или файла, внутри которого вы работаете.

С помощью метода `$wpdb->get_row($query, $output_type, $row_offset)` мы получаем только одну строку из результата. Вместо массива результатов вы получаете только первый объект (или массив, если указан тип `$output_type`) из набора результатов.

Вы можете указать параметр `$row_offset`, чтобы получить другую строку из результатов (0 — первая, 1 — вторая и т. д.).

Вставьте, замените и обновите. Метод `$wpdb->insert($table, $data, $format)` позволяет вставить данные в базу данных. Вместо того чтобы создавать собственный запрос `INSERT`, вы можете просто передать имя таблицы и ассоциативный массив, содержащий данные строки, и WordPress создаст запрос за вас. Ключи вашего массива `$data` должны соответствовать именам столбцов в таблице. Значения в массиве являются значениями для вставки в строку таблицы.

Пример

```

<?php
//обработка новых заданий для назначений
global $wpdb, $current_user;
// создать задание
$assignment_id = intval($_REQUEST['assignment_id']);
$submission_id = wp_insert_post(
    array(
        'post_type'    => 'submission',
        'post_author'  => $current_user->ID,
        'post_title'   => sanitize_title($_REQUEST['title']),
        'post_content' => sanitize_text_field($_POST['submission'])
    )
);
// связываем задачу и назначение
$wpdb->insert(
    $wpdb->schoolpress_assignment_submissions,
    array("assignment_id"=>$assignment_id, "submission_id"=>$submission_id),
    array('%d', '%d')
);

```

```

/*
    Этот вызов insert сгенерирует запрос SQL, например:
    INSERT INTO
    'wp_schoolpress_assignment_submissions'

    ('assignment_id', 'submission_id' VALUES (101,10)

*/
?>

```

В этом примере мы используем функцию `wp_insert_post()` для создания задания, а затем вызываем метод `$wpdb->insert()` для вставки строки в нашу пользовательскую таблицу, связывающую назначения с заданиями.

Мы передаем массив форматов третьему параметру, чтобы отформатировать данные как целые числа, при построении SQL-запроса. Доступные форматы: `%s` — для строк, `%d` — для целых чисел и `%f` — для чисел с плавающей точкой. Если формат не указан, все данные будут представлены в виде строки. В большинстве случаев MySQL будет правильно преобразовывать вашу строку в необходимый формат для ее сохранения в фактической таблице.

Чтобы связать две записи, подобные этой, мы также могли бы просто поместить `assignment_id` в столбец `post_parent` таблицы `wp_posts`. Этого достаточно для создания отношений родитель/ребенок. Однако если вы хотите установить связь "многие ко многим" (например, если вы можете опубликовать одну и ту же задачу в нескольких назначениях), то вам потребуется отдельная таблица или какой-либо другой способ связать сообщение со многими другими сообщениями.

Конструкция `$wpdb->replace($table, $data, $format)` аналогична `$wpdb->insert()`. Метод `$wpdb->replace()` буквально генерирует тот же SQL-запрос, что и `$wpdb->insert()`, но использует вместо команды `INSERT` команду MySQL `REPLACE`, отбрасывает любую строку с теми же ключами, что и передаваемые в `$data`.

Выражение `$wpdb->update($table, $data, $where, $format = null, $where_format = null)` позволяет обновлять строки в таблице базы данных. Вместо того чтобы создавать собственный запрос `UPDATE`, просто передайте таблицу и ассоциативный массив, содержащий обновленные столбцы и новые данные, вместе с ассоциативным массивом `$where`, содержащим поля для проверки в предложении `WHERE`, а WordPress создаст запрос и очистит его командой `UPDATE` для вас.

Параметры `$where` и `$where_format` работают так же, как массивы `$data` и `$format` соответственно.

Предложение `WHERE`, сгенерированное методом `update()`, проверит, что столбцы равны переданным значениям, и что эти проверки объединены вместе условиями `AND`.

Метод `update()` особенно полезен тем, что вы можете обновить любое количество полей в строке таблицы, используя одну функцию. Вот пример кода, который позволяет обновить заказы в плагине электронной коммерции.

Пример

```
<?php
global $wpdb;
// просто обновляем статус
$wpdb->update(
    'ecommerce_orders', // имя таблицы
    array('status' => 'paid'), // поля данных
    array('id' => $order_id) // поля where
);

// обновляем больше данных о заказе
$wpdb->update(
    'ecommerce_orders', // имя таблицы
    array('status' => 'pending', // поля данных
        'subtotal' => '100.00',
        'tax' => '6.00',
        'total' => '106.00'
    ),
    array('id' => $order_id) // поля where
);
?>

$wp_query
```

Объект класса `WP_Query`, который может показать вам весь контент публикации, возвращаемый WordPress для каждой страницы, на которой вы находитесь. Подробнее о классе `WP_Query` и его методах мы поговорим в следующей главе.

`$current_user` — объект всех данных, связанных с текущим вошедшим пользователем. Этот объект не только возвращает все данные текущего пользователя из таблицы `wp_users`, но также предоставляет вам роли и возможности текущего пользователя.

Пример

```
<?php
// приветствуем вошедшего в систему пользователя
global $current_user;
if (!empty($current_user->ID)) {
    echo 'Howdy, ' . $current_user->display_name;
}
?>
```

При написании кода для запуска на WordPress вы можете определить и использовать собственные глобальные переменные, если это имеет смысл. Это может избавить вас от необходимости переписывать код и вызывать функции, потому что, как только они определены, вы можете вызывать их снова и снова.

Бесплатные плагины

Есть несколько полезных бесплатных плагинов, которые могут помочь расширить ваше веб-приложение. Плагины существуют практически для любой цели, но если вы не нашли именно ту функциональность, которую ищете, то можете изменить существующий плагин (с открытым исходным кодом, верно?) Или создать совершенно новый, если вы готовы принять вызов.

Admin Columns

Admin Columns (bwawwp.com/admin-columns) — очень полезный плагин, который отлично подойдет для управления столбцами, отображаемыми в публикациях, комментариях, медиа и любых других зарегистрированных пользовательских типах записей на страницах. Этот плагин также поддерживает добавление пользовательских полей, избранных изображений и пользовательских таксономий в столбцы администратора. Особенно полезные функции этого плагина также включают возможность добавлять сортируемые столбцы, редактируемые поля и экспортируемые CSV-файлы с указанными вами столбцами. Профессиональная версия этого плагина также интегрируется с Advanced Custom Fields, WooCommerce и Yoast SEO, что позволяет вам добавлять любые метаполя из этих плагинов в столбцы администратора. Один из наших любимых вариантов использования этого плагина — возможность массового редактирования значений SEO для всех моих постов на странице вместо редактирования каждого отдельного поста; поговорим об экономии времени!

Advanced Custom Fields

Плагин Advanced Custom Fields (ACF) (bwawwp.com/acf) позволяет администраторам легко настраивать доступные настраиваемые поля для любого типа записей. Он имеет очень приятный пользовательский интерфейс администратора для установки полей и типов полей, доступных для данного типа сообщений, а внутренние пользователи могут взаимодействовать с установленными полями при добавлении или обновлении публикации. Этот плагин также предназначен для разработчиков, в нем множество пользовательских хуков и встроенных фильтров, а также пользовательских функций, которые служат для отображения данных в интерфейсе. ACF также имеет множество платных дополнительных плагинов, в том числе:

- ◆ Repeater field — позволяет вам создавать набор подполей, которые можно повторять снова и снова, как способ добавления списка значений.
- ◆ Gallery field — создает приятный интерфейс для управления несколькими изображениями.
- ◆ Options page — предоставляет функции для добавления дополнительных страниц администратора для обновления полей ACF.
- ◆ Flexible content field — создает пользовательские разметки с помощью UI контент-разметки.

BadgeOS

Плагин BadgeOS (bwawwp.com/badgeOS) может превратить любой веб-сайт в платформу для поощрения участников с учетом достижений, основанных на их деятельности. Администратор сайта может на его основе создать различные типы достижений и награждать участников общими значками, как только они выполнят все заявленные требования для определенного уровня. Значки совместимы с Mozilla OBI и доступны через Credly.com.

Еще один очень крутой плагин для геймификации WordPress — это GamiPress (bwawwp.com/gamipress), который на самом деле является ветвью BadgeOS. По нашему мнению, он лучше поддерживается и имеет больше возможностей.

Posts 2 Posts

Posts 2 Posts (P2P) (bwawwp.com/posts2posts) — еще один мощный плагин для создания веб-приложений. С его помощью вы сможете создавать отношения "многие ко многим" между постами, страницами и CPT, а также отношения "многие ко многим" между постами и пользователями.

Например, вы можете задействовать P2P для установления связи между CPT для школ, учителей и предметов. В школе может быть несколько учителей, и каждый учитель может быть привязан к одному или нескольким предметам.

Для создания новых связей P2P предоставляет интуитивно понятные настройки, многофункциональные виджеты и простой в использовании метаблок, прикрепленный к любой странице добавления/редактирования поста.

Разработчики пользовательских плагинов должны избегать создания дополнительных таблиц базы данных, если это не обусловлено крайней необходимостью. Если бы мы хотели связать посты с другими постами, то могли бы сохранить массив идентификаторов постов в настраиваемом поле другого поста, но такой подход может стать неэффективным в крупномасштабном приложении. P2P создает свои собственные таблицы базы данных, `wp_p2p` и `wp_p2pmeta`, для более эффективного хранения связей между публикациями (табл. 3.1 и 3.2 соответственно).

Таблица 3.1. Структура базы данных для таблицы `wp_p2p`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
<code>p2p_id</code>	<code>bigint(20)</code>		No	None	<code>AUTO_INCREMENT</code>
<code>p2p_from</code>	<code>bigint(20)</code>		No	None	
<code>p2p_to</code>	<code>bigint(20)</code>		No	None	
<code>p2p_type</code>	<code>varchar(44)</code>	<code>utf8_general_ci</code>	No		

Таблица 3.2. Структура базы данных для таблицы `wp_p2pmeta`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
<code>meta_id</code>	<code>bigint(20)</code>		No	None	AUTO_INCREMENT
<code>p2p_id</code>	<code>bigint(20)</code>		No	0	
<code>meta_key</code>	<code>varchar(255)</code>	<code>utf8_general_ci</code>	Yes	NULL	
<code>meta_value</code>	<code>longtext</code>	<code>utf8_general_ci</code>	Yes	NULL	

Для получения дополнительной информации об этом плагине обязательно ознакомьтесь с вики на GitHub (bawawp.com/p2p-wiki).

Members

Плагин **Members** (bawawp.com/members-wp) расширяет область вашего контроля над ролями и возможностями пользователя на вашем сайте. Это позволяет вам редактировать, а также создавать и удалять пользовательские роли и права. Этот плагин также позволяет вам устанавливать разрешения для различных пользовательских ролей, чтобы определить, какие пользователи могут добавлять, редактировать и/или удалять различные фрагменты содержимого. Хотя, изучив информацию из главы 6, можно написать пользовательский код для изменения ролей и разрешений, такой плагин, как **Members**, способен ускорить этот процесс для некоторых проектов, и он абсолютно необходим, если вам нужны непрограммисты для дальнейшей настройки указанных правил.

Плагин **User Role Editor** (oreil.ly/EA0my) также популярен для управления ролями и разрешениями через графический интерфейс.

W3 Total Cache

Кэширование контента — отличная идея для оптимизации производительности вашего сайта. Вы можете сэкономить много времени на обработку, отображая кэшированные страницы конечному пользователю, вместо того чтобы обращаться к базе данных каждый раз, когда кто-то запрашивает информацию. **W3 Total Cache** (bawawp.com/w3-cache) имеет множество встроенных функций для управления тем, что кэшируется и когда необходимо очищать кэш. Более подробные сведения об использовании **W3 Total Cache** можно найти в главе 14.

Yoast SEO

Yoast SEO (bawawp.com/yoast-seo) — это отличный плагин, который можно рекомендовать, если вы заинтересованы в поисковой оптимизации (SEO). После установки этот плагин автоматически оптимизирует ваш сайт для поисковых систем. Также добавлен предварительный просмотр фрагмента, отображаемого в результа-

тах поиска, с полями для переопределения заголовка и описания. Еще одна полезная функция плагина Yoast SEO — отчет "Анализ читабельности", содержащий предложения для облегчения понимания вашего текста.

Доступна также платная версия Yoast SEO с расширенными функциями и премиум-поддержкой.

Другой популярный плагин для решения SEO — All in One SEO Pack (bawawp.com/1-seo-pack).

Премиальные плагины

Хотя существует множество отличных бесплатных плагинов, некоторые премиальные плагины определенно стоят своих денег. Эти плагины обычно доступны для покупки для одноразового применения, но некоторые также предлагают лицензии разработчика, которые позволяют вам приобрести плагин для установки на нескольких сайтах WordPress.

Gravity Forms

Что очевидно из названия, плагин Gravity Forms (www.gravityforms.com) позволяет легко создавать пользовательские формы контактов для вашего сайта. Создание формы с помощью визуального редактора — чрезвычайно простой процесс, редактор позволяет перетаскивать поля в форму и перемещать их по мере необходимости. Включены стандартные поля, а также есть возможность создания пользовательских полей. Формы очень гибкие и могут быть настроены как многостраничные формы с индикаторами выполнения. Условные поля позволяют отображать или скрывать поля на основе выбора в предыдущих полях. Еще одна замечательная особенность этого плагина — возможность отправки форм после их заполнения в любое место, выбранное администратором сайта в настройках формы. В общем, этот плагин полезен и гибок для всех, кому нужно создать форму на своем сайте, и прост в использовании для тех, кто не имеет знаний в области программирования.

BackupBuddy

Плагин BackupBuddy (bawawp.com/backup-b) предоставляет вам возможность создать резервную копию всей вашей инсталляции WordPress для безопасного хранения, восстановления или перемещения вашего сайта. Резервное копирование может быть запланировано на регулярной основе, а затем файл может быть загружен на ваш компьютер, передан вам по электронной почте или отправлен в хранилище по вашему выбору, например Dropbox или FTP-сервер. Этот плагин также имеет функцию восстановления, которая легко восстановит ваши темы, виджеты и плагины. Плагин позволяет без проблем перемещать ваш сайт на новый сервер или домен прямо с панели управления WordPress, что очень удобно, если вы работаете на сервере Dev и затем при запуске перемещаете сайты в производственную среду.

WP All Import

Плагин WP All Import (www.wpallimport.com) пригодится, если вы хотите импортировать данные в WordPress из другого источника в формате XML или CSV, эти два формата WordPress обычно не воспринимает. Кроме того, существует профессиональная или премиум-версия плагина, доступная для покупки, которая расширяет функциональность, позволяя вам импортировать данные в CPT, а также в настраиваемые поля. Профессиональная версия позволяет импортировать изображения из URL-адреса и сохранять их в медиатеке. Также полезной является возможность настроить повторяющийся импорт, чтобы периодически проверять файл на наличие изменений или обновлений, а затем при необходимости изменять соответствующий пост.

Плагины сообщества

Вы можете создать полноценную социальную сеть с помощью WordPress и нескольких бесплатных плагинов. Социальные сети отлично подходят для объединения нишевого сообщества. Если у вас есть активная социальная сеть, значит, у вас будет много органического контента, индексируемого поисковыми системами.

Если вы думаете, что вы получаете много комментариев и взаимодействий на своем существующем веб-сайте WordPress, попробуйте превратить его в социальную сеть, чтобы получать реальные конверсии.

BuddyPress

BuddyPress (buddypress.org) — это социальная сеть "в коробке". Вы можете за несколько минут запустить социальную сеть с большинством функций Facebook.

BuddyPress можно скачать из хранилища плагинов, как и другие плагины, или с сайта BuddyPress. Этот плагин прошел долгий путь с версии 1.0, выпущенной в апреле 2009 года. Первоначально построенный Энди Питлингом, он работал только на WordPress MU (Multi User). Компания Automattic, видя потенциал плагина, который превращает WordPress в приложение для социальных сетей, начала финансировать проект.

Начиная с версии 1.7 BuddyPress перестал зависеть от темы, а значит, правильно запрограммированный, он будет работать практически с любой темой. До версии 1.7 (рис. 3.2), чтобы правильно использовать плагин, вы должны были назначить тему BuddyPress. Это хорошо работает для людей, желающих создать социальную сеть с нуля, потому что они могут взять встроенную по умолчанию тему, купить хорошую премиальную дочернюю тему BuddyPress или планировать создать собственную дочернюю тему BuddyPress. Однако это было ограничением для тех, у кого уже был веб-сайт WordPress, потому что они не могли просто включить BuddyPress и заставить его работать с исходной темой. В большинстве случаев людям с существующими сайтами, которые хотели включить BuddyPress, нужно было выполнить настройку, что хорошо для тех, кто знает CSS, PHP и то, как работает WordPress.

Но непрограммистам нужно было нанять кого-то, чтобы превратить существующую тему (за которую они, возможно, заплатили) в дочернюю или совместимую тему BuddyPress. Однако с более новыми версиями BuddyPress все работает без всяких проблем!

Люди с готовыми веб-сайтами теперь могут использовать BuddyPress и его функции и заставить его работать в своей существующей теме. Также очень легко переопределить любой имеющийся стиль, чтобы адаптировать функции BuddyPress к вашему веб-сайту. Особая благодарность более поздним основным разработчикам — Джону Джейкоби, Буну Горджесу, Полу Гиббсу и todo (настоящее имя Ray Ho): за создание BuddyPress, за то, что сегодня это независимый от темы плагин, который превращает WordPress в социальную сеть.

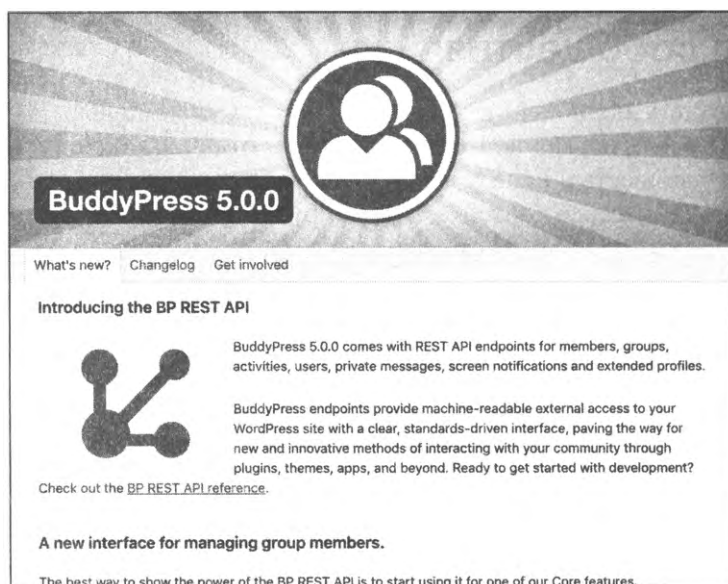


Рис. 3.2. Добро пожаловать в BuddyPress

Таблицы базы данных

В отличие от многих плагинов WordPress, BuddyPress создает свои собственные таблицы баз данных в MySQL. Если бы разработчики BuddyPress переписали плагин с нуля сегодня, они, вероятно, хранили бы мероприятия и уведомления в виде пользовательских записей, а не пользовательских таблиц. Тем не менее СРТ не были реализованы на момент появления первой версии BuddyPress, и потребуются много усилий, чтобы изменить эту архитектуру сейчас. Если в пользовательских таблицах хранятся группы пользователей и отношения дружбы между ними, то гораздо легче и быстрее сделать запрос, чем при хранении этих данных в виде некоторой комбинации сообщений, метапользователей и таксономий.

Для небольших распределенных плагинов имеет смысл избегать пользовательских таблиц всякий раз, когда это возможно, потому что тогда пользователям плагина

нужно беспокоиться о меньшем количестве вещей. Тем не менее для плагинов, характерных для вашего приложения, или плагинов, которые включают в себя столько функциональности, как BuddyPress, пользовательские таблицы могут помочь ускорить ваше приложение или лучше организовать данные. Мы включили структуру базы данных для каждой таблицы BuddyPress (табл. 3.3–3.18) в качестве примера того, как вы могли бы подойти к структурированию пользовательских таблиц для ваших собственных приложений, а также чтобы помочь вам понять, как данные хранятся в BuddyPress в случае, если вам захочется получить эту информацию напрямую.

Таблица 3.3. Структура базы данных для таблицы `wp_bp_activity`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
user_id	bigint(20)		No	None	
component	varchar(75)	utf8_general_ci	No	None	
type	varchar(75)	utf8_general_ci	No	None	
action	text	utf8_general_ci	No	None	
content	longtext	utf8_general_ci	No	None	
primary_link	varchar(255)	utf8_general_ci	No	None	
item_id	bigint(20)		No	None	
secondary_item_id	bigint(20)		Yes	NULL	
date_recorded	datetime		No	None	
hide_sitewide	tinyint(1)		Yes	0	
mptt_left	int(11)		No	0	
mptt_right	int(11)		No	0	
is_spam	tinyint(1)		No	0	

Таблица 3.4. Структура базы данных для таблицы `wp_bp_activity_meta`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
activity_id	bigint(20)		No	None	
meta_key	varchar(255)	utf8_general_ci	Yes	NULL	
meta_value	longtext	utf8_general_ci	Yes	NULL	

Таблица 3.5. Структура базы данных для таблицы `wp_bp_friends`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
initiator_user_id	bigint(20)		No	None	
friend_user_id	bigint(20)		No	None	
is_confirmed	tinyint(1)	utf8_general_ci	Yes	0	
is_limited	tinyint(1)		Yes	0	
date_created	datetime		No	None	

Таблица 3.6. Структура базы данных для таблицы `wp_bp_groups`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
creator_id	bigint(20)		No	None	
name	varchar(100)	utf8_general_ci	No	None	
slug	varchar(200)	utf8_general_ci	No	None	
description	longtext	utf8_general_ci	No	None	
status	varchar(100)	utf8_general_ci	No	Public	
enable_forum	tinyint(1)		No	1	
date_created	datetime		No	None	

Таблица 3.7. Структура базы данных для таблицы `wp_bp_groupmeta`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
group_id	bigint(20)		No	None	
meta_key	varchar(255)	utf8_general_ci	Yes	NULL	
meta_value	longtext	utf8_general_ci	Yes	NULL	

Таблица 3.8. Структура базы данных для таблицы `wp_bp_members`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
group_id	bigint(20)		No	None	
user_id	bigint(20)		No	None	
inviter_id	bigint(20)		No	None	
is_admin	tinyint(1)		No	0	
is_mod	tinyint(1)		No	0	
user_title	varchar(100)	utf8_general_ci	No	None	
date_modified	datetime		No	None	
comments	longtext	utf8_general_ci	No	None	
is_confirmed	tinyint(1)		No	0	
is_banned	tinyint(1)		No	0	
invite_sent	tinyint(1)		No	0	

Таблица 3.9. Структура базы данных для таблицы `wp_bp_messages_messages`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
thread_id	bigint(20)		No	None	
sender_id	bigint(20)		No	None	
subject	varchar(200)	utf8_general_ci	No	None	
message	longtext	utf8_general_ci	No	None	
date_sent	datetime		No	None	

Таблица 3.10. Структура базы данных для таблицы `wp_bp_messages_notices`

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
subject	bigint(20)		No	None	
message	bigint(20)	utf8_general_ci	No	None	

Таблица 3.10 (окончание)

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
date_sent	varchar(200)	utf8_general_ci	No	None	
is_active	longtext		No	None	

Таблица 3.11. Структура базы данных для таблицы wp_bp_messages_recipients

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
user_id	bigint(20)		No	None	
thread_id	bigint(20)		No	None	
unread_count	int(10)		No	0	
sender_only	tinyint(1)		No	0	
is_deleted	tinyint(1)		No	0	

Таблица 3.12. Структура базы данных для таблицы wp_bp_notifications

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
user_id	bigint(20)		No	None	
item_id	bigint(20)		No	None	
secondary_item_id	bigint(20)		Yes	NULL	
component_name	varchar(75)	utf8_general_ci	No	None	
component_action	varchar(75)	utf8_general_ci	No	None	
date_notified	datetime		No	None	
is_new	tinyint(1)		No	0	

Таблица 3.13. Структура базы данных для таблицы wp_user_blogs

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT

Таблица 3.13 (окончание)

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
user_id	bigint(20)		No	None	
blog_id	vbigint(20)		No	None	

Таблица 3.14. Структура базы данных для таблицы wp_user_blogmeta

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
blog_id	bigint(20)		No	None	
meta_key	varchar(255)	utf8_general_ci	Yes	NULL	
meta_value	longtext	utf8_general_ci	Yes	NULL	

Таблица 3.15. Структура базы данных для таблицы wp_bp_xprofile_data

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
field_id	bigint(20)		No	None	
user_id	bigint(20)		No	None	
value	longtext	utf8_general_ci	No	None	
last_updated	datetime		No	None	

Таблица 3.16. Структура базы данных для таблицы wp_bp_xprofile_fields

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
group_id	bigint(20)		No	None	
parent_id	bigint(20)		No	None	
type	varchar(150)	utf8_general_ci	No	None	
name	varchar(150)	utf8_general_ci	No	None	
description	longtext	utf8_general_ci	No	None	

Таблица 3.16 (окончание)

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
is_required	tinyint(1)		No	0	
is_default_option	tinyint(1)		No	0	
field_order	bigint(20)		No	0	
option_order	bigint(20)		No	0	
order_by	varchar(15)	utf8_general_ci	No		
can_delete	tinyint(1)		No	1	

Таблица 3.17. Структура базы данных для таблицы wp_bp_xprofile_groups

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
name	varchar(150)	utf8_general_ci	No	None	
description	mediumtext	utf8_general_ci	No	None	
group_order	bigint(20)		No	0	
can_delete	tinyint(1)		No	None	

Таблица 3.18. Структура базы данных для таблицы wp_bp_xprofile_meta

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		No	None	AUTO_INCREMENT
object_id	bigint(20)		No	None	
object_type	varchar(150)	utf8_general_ci	No	None	
meta_key	varchar(255)	utf8_general_ci	Yes	NULL	
meta_value	longtext	utf8_general_ci	Yes	NULL	

Компоненты

После активации BuddyPress, над панелью компонентов (рис. 3.3) на вкладке **Settings** → **BuddyPress** или в файле `/wp-admin/options-general.php?page=bp-components` можно настроить набор компонентов, которые вы хотели бы использовать.

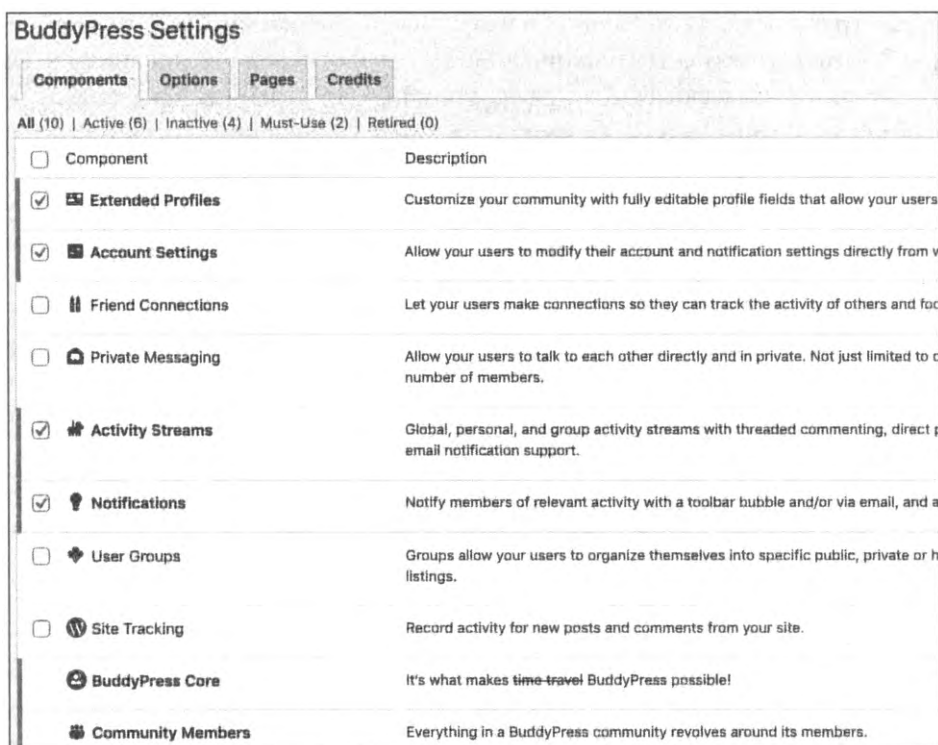


Рис. 3.3. Компоненты BuddyPress

Доступны следующие варианты:

- ◆ *Расширенные профили* — как и любая типичная социальная сеть, BuddyPress имеет профили пользователей. Участник обладает полным контролем над своим профилем. По умолчанию все участники перечислены в каталоге участников; когда вы нажимаете на имя пользователя, вы попадаете на страницу его профиля.
- ◆ *Настройки учетной записи* — участники могут обновить свой адрес электронной почты, изменить свой пароль и даже управлять уведомлениями по электронной почте, которые они получают, когда другие участники взаимодействуют с ними.
- ◆ *Друзья* — участники могут добавлять друг друга в друзья. Участники находят новых друзей, отправляя и получая запросы на добавление в друзья. Ровно как друзья Facebook.
- ◆ *Личные сообщения* — участники отправляют личные сообщения друг другу и просматривают свои сообщения в одном месте, например во входящих в вашей социальной сети. Участники могут отвечать, помечать как прочитанные, удалять и выполнять другие действия с сообщениями, как в любой большой социальной сети.
- ◆ *Потоки активности* — участники могут публиковать обновления действий в своих профилях и группах, оставлять комментарии о действиях других участни-

ков или групп, а также добавлять в избранные публикации. Похоже на Facebook, верно? У BuddyPress есть функция @mention, аналогичная упоминаниям в Twitter. @mention автоматически ссылается на упомянутую страницу профиля пользователя, и если у этого участника включены уведомления, он получает электронное письмо. Активность также входит в стандартную RSS-ленту.

- ♦ *Группы пользователей* — мощный компонент BuddyPress, группы могут создаваться органически (или нет) членами сети. Каждая группа появляется на странице со списком групп, и, нажав на аватар группы, вы попадаете на ее страницу. Профиль группы настраивается так же, как страница профиля пользователя, но с подстраницами, специфичными для группы, такими как групповые действия, члены, настройки администратора и приглашение друзей. Группы могут быть общедоступными, частными или скрытыми, а участники могут быть повышены до администраторов или модераторов группы.
- ♦ *Отслеживание сайта* — новые сообщения и комментарии на вашем сайте создают сообщения активности BuddyPress. Если вы используете BuddyPress в многосайтовой сети WordPress, посты и комментарии, созданные на любом сайте вашей сети, также будут создавать посты активности BuddyPress.

Все эти основные компоненты BuddyPress могут быть расширены с помощью плагинов BuddyPress. Это может показаться сложным, если вы новичок в данном вопросе, но вы можете установить дополнительные плагины, специфичные для BuddyPress, или создать свой собственный. Существует около 485 плагинов WordPress, которые так или иначе расширяют BuddyPress или интегрируются с ней.

Страницы

Как только вы определились, какие основные компоненты вам нужны, перейдите на вкладку **Pages (Settings → BuddyPress → Pages)**, как показано на рис. 3.4.

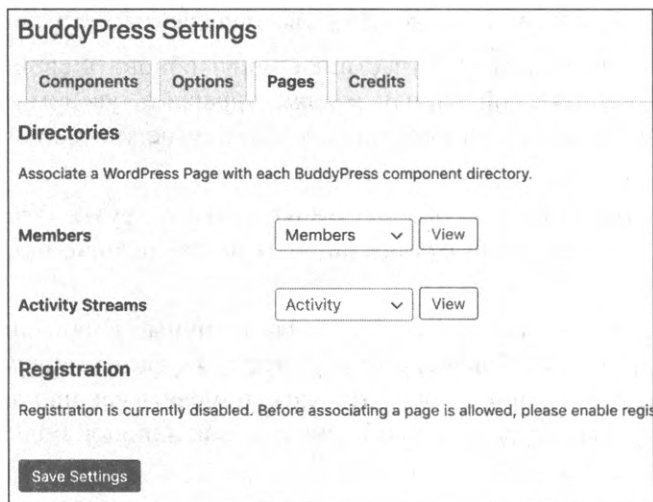


Рис. 3.4. Страницы BuddyPress

BuddyPress отображает используемые компоненты на новые или существующие страницы. По умолчанию BuddyPress пытается создать новую страницу для каждого компонента. Если вы хотите называть участников "Студенты" вместо "Участники", то можете создать обычную страницу WordPress под названием "Студенты" и сопоставить компонент участников с этой новой страницей. То же самое относится и к другим композициям BuddyPress. Вам необходимо создать две страницы для регистрации участника: Регистрация и Активация. Вам нужно будет сопоставить оба этих элемента со страницами, если вы хотите иметь открытую регистрацию в своей социальной сети.



Чтобы разрешить открытую регистрацию, вы также должны убедиться, что каждый может зарегистрироваться; установите флажок **Anyone can register** в разделе **Settings** → **General**.

Настройки

На панели **Settings** (**Settings** → **BuddyPress** → **Settings**) или странице `/wp-admin/admin.php?page=bp-settings` вы можете настроить некоторые дополнительные параметры BuddyPress:

- ♦ Панель инструментов — по умолчанию BuddyPress показывает админ-панель WordPress со ссылками **Login** и **Register** для пользователей, которые не вошли в систему. Здесь вы также можете отключить эту функцию.
- ♦ Удаление аккаунта — вы можете решить, разрешите ли вы зарегистрированным пользователям удалять свои учетные записи.
- ♦ Аватар для загрузки — вы можете разрешить зарегистрированным пользователям загружать аватары.
- ♦ Синхронизация профиля — включить синхронизацию BuddyPress с профилем WordPress (BuddyPress->WordPress).
- ♦ Создание группы — вы можете разрешить зарегистрированным пользователям создавать свои собственные группы. Администраторы сайта могут создавать группы, даже если этот параметр отключен.
- ♦ Комментарии к блогам и форумам — вы можете разрешить комментирование потока активности в блоге и на форуме.

Поля профиля

Эта функция BuddyPress, расположенная в разделе **Users** → **Profile Fields**, позволяет создавать любое количество групп полей профиля и полей профиля для ваших участников. Вы можете собирать данные, такие как местоположение, дата рождения, лайки, антипатии, любимый цвет и/или все, что хотите. Эта функция очень гибкая и позволяет вам организовывать поля вашего профиля в разные группы профилей, которые будут доступны на странице профиля любого пользователя.

При добавлении любого нового поля профиля вам предоставляется удобный интерфейс для принятия решения о том, нужно ли делать новое поле обязательным, какой выбрать тип элемента формы, какова видимость по умолчанию и хотите ли вы, чтобы ваши пользователи могли сами изменить видимость поля. Форма показана на рис. 3.5.



По умолчанию все поля профиля в группе профилей "База" отображаются на странице регистрации.

Рис. 3.5. Поля профиля BuddyPress

Плагины для BuddyPress

Как видите, BuddyPress — это интуитивно понятный и простой в использовании плагин. Мы кратко поговорили об установке дополнительных плагинов, специфичных для BuddyPress, и далее приведем краткий список некоторых классных плагинов, чтобы дать вам идеи того, как вы можете расширить BuddyPress:

- ♦ **BuddyPress Media (bawwp.com/bp-rtmedia)** — этот плагин позволяет вашим пользователям загружать фотографии, музыку и видео в свои посты. Это также дает возможность вашим пользователям организовать все свои фотографии в фотоальбомы на странице своего профиля. Существует поддержка мобильных устройств, которая включает в себя автоматическое преобразование аудио и видео.
- ♦ **BuddyPress Registration Options (bawwp.com/bp-regis)** — это отличный плагин для предотвращения регистрации спам-ботов на вашем сайте BuddyPress. Пла-

гин позволяет настроить модерацию новых пользователей. Если модерация включена на странице настроек администратора, то любые новые участники будут заблокированы от взаимодействия с любыми компонентами BuddyPress (кроме редактирования своего профиля и загрузки своего аватара) и не будут перечислены в каком-либо каталоге до того, как администратор утвердит или отклонит их учетную запись. Если модерация включена, то администраторы могут создавать настраиваемые отображаемые сообщения и оповещения по электронной почте для утвержденных или отклоненных учетных записей. Когда администратор одобряет или отклоняет, электронные письма отправляются новым пользователям, информируя их, была ли их регистрация одобрена.

- ◆ *Надстройка AppPresser (bwawwp.com/apbp)* — приложение AppPresser AppCommunity позволяет пользователям AppPresser создать социальное мобильное приложение для своего веб-сайта WordPress, аналогичное Instagram или Facebook, с помощью BuddyPress. С AppCommunity пользователи приложения могут публиковать обновления активности и фотографии, создавать дружеские отношения, отправлять личные сообщения, присоединяться к группам и многое другое.
- ◆ *Надстройка BadgeOS Community (bwawwp.com/badgeos)* — надстройка BadgeOS Community интегрирует функции BadgeOS в BuddyPress и bbPress. Участники сайта завершают достижения и получают значки, основываясь на ряде общих мероприятий и других действий. Это дополнение к BadgeOS также включает возможность отображения значков и достижений в профилях пользователей и каналах активности.
- ◆ *bbPress (bbpress.org)* — если есть форумы, то bbPress может удовлетворить все ваши потребности в форуме. В отличие от BuddyPress, bbPress использует пользовательские типы записей, поэтому он не создает свои собственные таблицы в базе данных, как это было в предыдущих версиях.

Применение bbPress может потребовать небольшой работы с темой, если ваша тема еще не разработана для поддержки bbPress, но это, безусловно, самый простой способ добавить функциональность форума на сайт WordPress.

Темы WordPress — это фронтенд вашего веб-приложения. В *главе 1* мы привели аналогию, что темы WordPress похожи на представления в традиционной среде MVC. Аналогия не идеальна, но темы и представления схожи в том, что и те и другие управляют внешним видом вашего приложения, чем большую часть времени будут заниматься ваши дизайнеры.

Справочник разработчика тем (oreil.ly/CN56I), составленный сообществом WordPress — это довольно полный источник информации о том, как создавать темы для WordPress, основываясь на стандартах. Все разработчики тем должны использовать этот ресурс. В этой главе рассматриваются аспекты создания тем, особенно важные для разработчиков приложений.

Темы и плагины

С точки зрения разработчика все исходные файлы в ваших темах и плагинах — это всего лишь PHP-файлы, загружаемые WordPress в разное время. Теоретически весь программный код вашего приложения может находиться в одной теме или в одном плагине. На практике вы захотите зарезервировать свою тему для кода, связанного с внешним интерфейсом (представлениями) вашего веб-сайта, и использовать плагины для бэкенда вашего приложения (модели и контроллеры).

Место, куда вы решите поместить какой-то код, в основном будет зависеть от того, будете ли вы собирать полное приложение или отдельный плагин или тему.

Где разместить код при разработке приложений

Если вы создаете полноценное веб-приложение — по факту одну инсталляцию WordPress — у вас будет полный доступ к сайту и темам, которые установлены. Ваш код можно поместить куда угодно. Тем не менее вы должны предварительно продумать все нюансы при принятии решения о том, должна ли конкретная функция быть закодирована как модуль плагина или темы вашего приложения или как отдельный плагин. Преимущество от хорошего планирования на этом этапе получают в основном ваши разработчики (может быть, только вы). Правильная организация вашего программного кода облегчит поддержку вашего приложения и дальнейшее его развитие.

При создании приложений мы стараемся следовать таким рекомендациям:

- ♦ Используйте один главный плагин для хранения основного кода приложения и одну тему для управления кодом внешнего интерфейса.

- ◆ Любая модульная функциональность, которая может быть полезна в других проектах или может быть заменена другим плагином, должна быть закодирована в виде отдельного плагина.
- ◆ Никогда не вмешивайтесь в код ядра!¹

Итак, что такое основной код приложения и что такое код внешнего интерфейса? Опять же, наш псевдоMVC фреймворк выглядит так:

Плагины = модели.

Все ваши структуры данных, определяющие код, бизнес-логика и службы Аях должны быть включены в основной плагин. Такие вещи, как определения СРТ и таксономии, обработка форм и обертки для классов `Post` и `User` также должны войти в ваш основной плагин.

Темы = представления.

Весь ваш код шаблонов и логика внешнего интерфейса должны находиться в вашей теме. Фрейм вашего сайта, верхний и нижний колонтитулы, меню и боковые панели должны обрабатываться в вашей теме. Простая логика вроде `if(is_user_logged_in()) { //show menu } else { //show login }` тоже должна войти в код вашей темы.

Когда вы решаете, где писать функции, то нужно учитывать состав и возможности вашей команды разработчиков. Если у вас есть дизайнер и программист, вы должны поместить в тему то, о чем будет думать дизайнер, а все, что касается программиста, — в основной плагин. Даже если вам придется немного тут поработать головой, четкое разделение подобных вещей упростит жизнь вашим разработчикам.

Где разместить код при разработке плагинов

Если вы создаете плагин для использования на других веб-сайтах или модульные функции, которые могут применяться в разных проектах, имеет смысл сохранить код в отдельном плагине. В этих случаях вы можете хранить файлы шаблонов в вашем плагине для обработки компонентов пользовательского интерфейса. Обычная практика — перезапись этих файлов активной темой WordPress, о чем будет рассказано далее в этой главе.

Где разместить код при разработке тем

Точно так же, если вы разрабатываете тему, предназначенную для последующего распространения и опирающуюся на СРТ или другие настройки, которые обычно кодируются в плагине, возможно, имеет смысл вместо этого включить такой код в вашу тему. Если ваши пользователи должны активировать плагин, прежде чем ва-

¹ Если вы обнаружите, что вам необходимо изменить код основных файлов, чтобы заставить что-то работать, сначала еще раз проверьте, действительно ли вам нужно это сделать. Если да, вам требуется изменить основной файл WordPress, то вместо этого лучше добавить хуки и отправить их как патч для следующей версии WordPress.

ша тема вообще будет работать, вы также можете переместить код плагина в вашу тему. Если ваша тема вносит большие изменения в WordPress, подумайте о том, чтобы вставить код "плагина" в родительскую тему, а код, связанный с дизайном, — в дочернюю тему. Таким образом, если ваши пользователи захотят изменить дизайн своего сайта без потери других функций, предоставляемых темой, они смогут сделать это проще.

С другой стороны, если код, который вы собираетесь добавить в свою тему, не имеет решающего значения для работы темы или есть другие плагины, которые могли бы использоваться в качестве альтернативы вашему коду, то вы должны переместить этот код в плагин и распространять вашу тему как пакет, который включает в себя темы и рекомендуемые плагины. Например, многие темы премиум-класса добавляют связанные с SEO поля на страницу редактирования сообщений для управления заголовками страниц, метаописаниями и метаключевыми словами. Это имеет смысл, поскольку подобные поля, связанные с SEO, видят Google и другие веб-сканеры. Как бы то ни было, есть несколько действительно популярных плагинов, которые выполняют ту же функцию, и трудно утверждать, что ваша тема не будет работать без установленной функциональности SEO. Мы бы порекомендовали разработчикам тем помещать свои функции SEO в плагины или иным образом облегчить их отключение, чтобы можно было использовать другие плагины.

В конце концов, решение о том, куда какой код помещать и как все это упаковывать, должно приниматься с учетом ваших потребителей, как конечных пользователей, так и разработчиков, которые будут использовать ваши темы и плагины. Одно из достоинств WordPress — его гибкость с точки зрения настройки. Строгих правил нет. Рассматривайте все, что вы читаете по этой теме (в том числе в данной книге), как руководство. Если перемещение программного кода из файла плагина в файл темы облегчает работу, сделайте это.

Иерархия шаблонов

Когда пользователь посещает ваш сайт и переходит на какую-то страницу, WordPress задействует систему, называемую *иерархией шаблонов*, чтобы определить, какой файл в активной теме следует использовать для визуализации страницы. Например, если пользователь переходит на страницу с одним постом, WordPress будет искать файл `single-post.php`. Если данный файл не найден, начнется поиск файла `single.php`. Если и этот файл не найден, то следующим для поиска будет `index.php`.

Файл `index.php` — это запасной вариант для всех страниц, а вместе с файлом `style.css` — единственный файл, необходимый для вашей темы. Как правило, у вас будет примерно такой список файлов: `404.php`, `author.php`, `archive.php`, `attachment.php`, `category.php`, `comments.php`, `date.php`, `footer.php`, `front-page.php`, `functions.php`, `header.php`, `home.php`, `image.php`, `index.php`, `page.php`, `search.php`, `sidebar.php`, `single.php`, `single-(post-type).php`, `style.css`, `tag.php`, `taxonomy.php`.

Некоторые файлы в этом списке загружаются при вызове определенной функции `get`. Например, `get_header()` загружает `header.php`, `get_footer()` — `footer.php`,

а `get_sidebar()` — `sidebar.php`. Передача параметра `name` этим функциям добавит загрузку файлов по имени; так, например, `get_header('alternate');` загрузит `header-alternate.php` из папки темы.

Функция `comments_template()` загружает `comments.php`, если в качестве первого параметра не указано другое имя файла.

Функция `get_search_form()` ищет файл `searchform.php` в папке вашей темы или выводит форму поиска WordPress по умолчанию, если файл не найден.

В документации по иерархии шаблонов WordPress (bawawwp.com/temp-hier) четко определены все файлы, которые WordPress будет искать в папке темы при загрузке страницы. Вы также можете посмотреть [Twenty Nineteen Theme \(bawawwp.com/2019-theme\)](http://bawawwp.com/2019-theme) или другую хорошо написанную тему, чтобы увидеть, какие имена файлов WordPress будет искать. Прочитайте комментарии в этих темах, чтобы понять, когда загружается та или иная страница.

При разработке приложений с пользовательскими типами записей обычно требуется применить другой шаблон при просмотре ваших типов записей в веб-интерфейсе. Вы можете переопределить вид одного поста или архива для ваших типов записей, добавив файлы с именами `single-<post_type>.php` и `archive-<post_type>.php`, где для `post_type` установлено значение, созданное при регистрации нового типа записи.

Шаблоны страниц

Один из самых простых способов запустить произвольный код PHP на веб-сайте WordPress — создать шаблон страницы в своей теме, а затем использовать этот шаблон на одной из ваших страниц.

Некоторые распространенные шаблоны в темах WordPress включают в себя контактные формы и формы целевой страницы (лендингов).

Образец шаблона страницы

Листинг 4.1 — это урезанная версия шаблона формы обратной связи, которую вы можете поместить в папку вашей темы.

Листинг 4.1. Образец шаблона страницы

```
<?php
/*
Template Name: Page - Contact Form
*/

// Получить значения, возможно, отправленные в форме
$email = sanitize_email($_POST['email']);
$name = sanitize_text_field($_POST['cname']);
```

```

$phone = sanitize_text_field($_POST['phone']);
$message = sanitize_text_field($_POST['message']);
$sendemail = !empty($_POST['sendemail']);

// форма отправлена?
if (!empty($sendemail)
    && !empty($cname)
    && !empty($email)
    && empty($lname)) {

    $mailto = get_bloginfo('admin_email');
    $mailsubj = "Contact Form Submission from " . get_bloginfo('name');
    $mailhead = "From: " . $cname . " <" . $email . ">\n";
    $mailbody = "Name: " . $cname . "\n\n";
    $mailbody .= "Email: $email\n\n";
    $mailbody .= "Phone: $phone\n\n";
    $mailbody .= "Message:\n" . $message;

    // отправить нам письмо
    wp_mail($mailto, $mailsubj, $mailbody, $mailhead);

    // установить сообщение для этой страницы и очистить переменные
    $msg = "Your message has been sent.";

    $email = "";
    $cname = "";
    $phone = "";
    $message = "";
}

elseif (!empty($sendemail) && !is_email($email))
    $msg = "Please enter a valid email address.";
elseif (!empty($lname))
    $msg = "Are you a spammer?";
elseif (!empty($sendemail) && empty($cname))
    $msg = "Please enter your name.";
elseif (!empty($sendemail) && !empty($cname) && empty($email))
    $msg = "Please enter your email address.";

// загружаем шапку страницы
get_header();
?>

<div id="wrapper">
    <div id="content">
        <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
            <h1><?php the_title(); ?></h1>
            <?php if (!empty($msg)) { ?>

```

```

    <div class="message"><?php echo $msg?></div>
<?php } ??>
<form class="general" action="<?php the_permalink(); ?>" method="post">
    <div class="form-row">
        <label for="cname">Name</label>
        <input type="text" name="cname" value="<?php echo esc_attr($cname);?>" />
        <small class="red">* Required</small>
    </div>
    <div class="hidden">
        <label for="lname">Last Name</label>
        <input type="text" name="lname" value="<?php echo esc_attr($lname);?>" />
        <small class="red">LEAVE THIS FIELD BLANK</small>
    </div>
    <div class="form-row">
        <label for="email">Email</label>
        <input type="text" name="email" value="<?php echo esc_attr($email);?>" />
        <small class="red">* Required</small>
    </div>
    <div class="form-row">
        <label for="phone">Phone</label>
        <input type="text" name="phone" value="<?php echo esc_attr($phone);?>" />
    </div>
    <div class="form-row">
        <label for="message">Question or Comment</label>
        <textarea class="textarea" id="message" name="message" rows="4" cols="55">
            <?php echo esc_textarea($message)?>
        </textarea>
    </div>

    <div class="form-row">
        <label for="sendemail">&nbsp;</label>
        <input type="submit" id="sendemail" name="sendemail" value="Submit" />
    </div>
</form>
<?php endwhile; endif; ?>
</div>
<?php
// загружаем "подвал" страницы
get_footer();
?>

```

WordPress будет сканировать все PHP-файлы в папке и подпапках вашей активной темы (а также в папке и вложенных папках родительской темы) в поиске шаблонов. Любые файлы, найденные с комментарием, включающим фразу "Template Name:", будут доступны в качестве шаблона.

Шаблон загружается после запуска WordPress командой `init` и того как выполнятся события `wp`. Заголовок темы и событие `wp_head` не будут загружены, пока вы не вызовете функцию `get_header()` в своем шаблоне. Таким образом, вы можете использовать верхнюю часть файла шаблона для обработки ввода формы и возможного перенаправления перед отправкой любых заголовков на страницу.

Ваш файл шаблона должен включать ту же разметку HTML, что и файл `page.php` вашей темы или шаблон одного поста. В предыдущем примере мы создаем обертку и содержимое `<div>` вокруг содержания формы контакта.

В предыдущем коде также имеются функции `sanitize_text_field()` и `sanitize_email()` для очистки значений, представленных формой. Функции `esc_attr()` и `esc_textarea()` служат для предотвращения атак межсайтового выполнения сценариев (XSS). Эти функции описаны в *главе 8*.

В контактной форме из предыдущего примера есть еще "приманка". Поле с именем `lname` будет скрыто с помощью CSS, поэтому обычные пользователи не увидят это поле и, следовательно, оставят его пустым при отправке формы. Боты, которые хотят воспользоваться вашей контактной формой для рассылки спама, увидят поле `lname` и внесут в него какое-то значение. Код, который обрабатывает форму, проверяет, что поле `lname` пустое, а затем отправляет электронное письмо. Подобно тому, как горшок с медом привлекает насекомых, скрытое поле `lname` привлекает спамеров, поэтому вы не отправляете электронную почту от их имени.

Использование хуков для копирования шаблонов

Если вы не хотите менять несколько файлов шаблонов при обновлении идентификаторов или имен классов ваших оберток `<div>`, то можете создать шаблон, который использует фильтр `the_content` или другое событие, специфичное для вашей темы, чтобы поместить контент в область основного контента вашей страницы. Затем вы можете загрузить другой файл шаблона, например основной шаблон `page.php`, который будет включать вызовы для загрузки фрейма вашего сайта и макета по умолчанию. В листинге 4.2 показано, как создать шаблон страницы, который загружает шаблон `page.php` и добавляет дополнительное содержимое под ним на определенных страницах.

Листинг 4.2. Шаблон с хуками

```
<?php
/*
    Template Name: Hooking Template Example
*/

// использовать шаблон страницы по умолчанию
require_once(dirname(__FILE__) . "/page.php");

// теперь добавляем контент, используя функцию,
// вызываемую во время выполнения хука the_content
```

```

function template_content($content)
{
    // получаем текущий пост в этом цикле
    global $post;

    // получаем объект post для текущей страницы
    $queried_object = get_queried_object();

    // мы не хотим фильтровать посты, которые не являются основным постом
    if(empty($queried_object) || $queried_object->ID != $post->ID)
        return $content;

    // захватить вывод
    ob_start();
    ?>
    <p>This content will show up under the page content.</p>
    <?php
    $temp_content = ob_get_contents();
    ob_end_clean();

    // добавляем и возвращаем содержимое шаблона
    return $content . $temp_content;
}

add_action("the_content", "template_content");

```

В листинге 4.2 применена небольшая хитрость, чтобы сравнить текущую запись `$post` с `$queried_object`. Как правило, глобальная переменная `$post` будет основной публикацией страницы, на которую вы перешли. Однако другие циклы на вашей странице будут временно устанавливать глобальные значения `$post` на любой пост, с которым они имеют дело в данный момент. Например, если ваш шаблон использует меню WordPress, то в действительности это будет цикл по записям типа меню. Многие боковые панели и разделы нижнего колонтитула будут проходить через другие наборы записей.

Функция `get_queried_object()` возвращает основной объект публикации для текущей страницы. Функция возвращает другой, но соответствующий объект, если вы находитесь на странице термина или автора. Функция возвращает ноль на страницах архива. Поскольку предыдущий пример — это шаблон страницы, который будет загружен только при просмотре одной страницы, вызов `get_queried_object()` всегда будет возвращать объект `$post` для текущей страницы.

Вы также можете вставить свой собственный хук в ваш файл `page.php` и другие основные шаблоны, чтобы сделать что-то подобное. Просто добавьте что-то вроде `do_action('my_template_hook');` в том месте вашего шаблона, где вы хотите вставить дополнительный контент.

Когда следует использовать шаблон темы?

В главе 3 мы рассмотрели, как использовать *шорткоды* для создания страниц для ваших плагинов. Шорткоды полезны, потому что они позволяют вам добавлять контент, управляемый CMS, выше и ниже шорткода в поле содержимого поста и оставляют код сосредоточенным в вашем плагине. Итак, если вы распространяете плагин и вам нужен этот шаблон страницы в комплекте, то для генерации вашей страницы предпочтительнее метод шорткода.

Точно так же, если вы распространяете тему самостоятельно, вам нужно будет включить любые шаблоны, необходимые для темы, в папку темы. Вы можете включить код для шаблонов, основанных на шорткодах, в свою тему, но более стандартный способ — создание шаблонов страниц.

И наконец, если вашему шаблону нужно изменить HTML-код макетов страниц по умолчанию, вы захотите использовать файл шаблона внутри вашей темы. Код листинга 4.2 прикрепляется к шаблону `page.php`, чтобы избежать необходимости перезаписывать HTML-код. Но если весь смысл шаблона заключается в переписывании HTML-кода (например, с помощью шаблона целевой страницы, в котором вы хотите скрыть заголовок, нижний колонтитул и меню по умолчанию), то вам определенно придется прибегнуть к шаблонам.

Функции WordPress для работы с темами

Далее мы обсудим функцию `get_template_part($slug, $name = null)`, которую можно использовать для загрузки других файлов `*.php` (части шаблона) в файл вашей темы.

Согласно Кодексу, `$slug` означает "имя фрагмента URL для общего шаблона", а `$name` относится к "названию специализированного шаблона". В действительности оба параметра просто объединяются дефисом для формирования искомого имени файла: `slug-name.php`.

Тема Twenty Twelve использует функцию `get_template_part()` для загрузки части "содержимого" поста определенного формата в цикл WordPress:

```
<?php /* Start the Loop */ ?>
<?php while (have_posts()) : the_post(); ?>
    <?php get_template_part('content', get_post_format()); ?>
<?php endwhile; ?>
```

Если ваша часть шаблона находится в подпапке вашей темы, добавьте имя папки в начале слага:

```
get_template_part('templates/content', 'page');
```

Функция `get_template_part()` использует функцию `locate_template()` WordPress, чтобы найти указанную часть шаблона, которая затем загружает файл с помощью функции `load_template()`. Функция `locate_template()` сначала ищет в дочерней теме. Если соответствующий файл не найден в дочерней теме, выполняется поиск в родительской теме.

Помимо поиска файла в дочерних и родительских темах, другим преимуществом применения `get_template_part()` по сравнению со стандартным вызовом PHP `include` или `require` является то, что перед включением файла устанавливается набор глобальных переменных WordPress. Следующий пример — исходный код функции `load_template()` из WordPress 4.9.7², показывающий установленные глобальные переменные. Обратите внимание, что массив `query_vars` также извлекается в локальную область. Аргументу `query_vars $s` уделяется особое внимание, и здесь он обрабатывается специальным образом (`escape`), поскольку он является распространенным вектором для атак XSS, и многие темы забывают экранировать поисковый запрос при отображении его в поле поиска.

Пример

```
<?php
function load_template($template_file, $require_once = true) {
    global $posts, $post, $wp_did_header, $wp_query, $wp_rewrite;
    global $wpdb, $wp_version, $wp, $id, $comment, $user_ID;

    if (is_array($wp_query->query_vars))
        extract($wp_query->query_vars, EXTR_SKIP);

    if (isset($s))
        $s = esc_attr($s);

    if ($require_once)
        require_once($template_file);
    else
        require($template_file);
}
?>
```

Использование переменной `locate_template` в плагинах

Обычный подход при проектировании плагинов — это включение шаблонов в папку плагинов и предоставление пользователям возможности переопределять эти шаблоны, добавляя свои собственные версии в активную тему. Например, в SchoolPress учителя могут приглашать учеников в свой класс. Форма приглашения хранится в шаблоне в плагине, как показано в следующем примере.

² Мы добавили разрыв строки и удалили несколько фигурных скобок, чтобы код лучше смотрелся в книге.

Пример

```
//schoolpress/templates/invite-students.php
?>
<p>Enter</p>
<form action="" method="post">
    <label for="email">Email:</label>
<input type="text" id="email" name="email" value="" />
    <input type="submit" name="invite" value="Invite Student" />
</form>
```

SchoolPress рассматривается как приложение SaaS, но мы также планируем выпустить версию плагина, которую другие смогут использовать на своих сайтах. Пользователи плагина могут захотеть переопределить шаблон по умолчанию, не редактируя основной плагин, поскольку изменения в основном плагине будут перезаписаны при обновлении плагина.

Чтобы позволить пользователям нашего плагина переопределить шаблон приглашения, мы создаем код, приведенный в следующем примере, при включении файла шаблона.

Пример

```
//schoolpress/shortcodes/invite-students.php
function sp_invite_students_shortcode($atts, $content=null, $code="")
{
    // начать буферизацию вывода
    ob_start();

    // ищем часть приглашения для студентов в активной теме
    $template = locate_template('schoolpress/templates/invite-students.php');

    // если не найден, использовать значение по умолчанию
    if(empty($template))
        $template = dirname(__FILE__) .
            '../templates/invite-students.php';

    // загрузить шаблон
    load_template($template);

    // получить содержимое из буфера и вывести его
    $temp_content = ob_get_contents();
    ob_end_clean();
    return $temp_content;
}
add_shortcode('invite-students', 'sp_invite_students_shortcode');
```

Этот код использует наш шаблон шорткода из *главы 3*. Но вместо того, чтобы встраивать HTML в функцию шорткода, мы загружаем его из файла шаблона. Сначала мы вызываем функцию `locate_template()` для поиска шаблона в активных дочерних и родительских темах, а затем, если файл не найден, мы устанавливаем значение переменной `$template` как путь шаблона по умолчанию, связанный с плагином. Шаблон загружается с помощью `load_template()`.

Файл style.css

Файл `style.css` вашей темы должен содержать комментарий, используемый WordPress для отслеживания версии темы, и другую информацию, отображаемую на информационной панели WordPress. В следующем примере приведен комментарий из верхней части файла `style.css` в теме "Twenty Nineteen".

Пример

```
/*
Theme Name: Twenty Nineteen
Theme URI: https://wordpress.org/themes/twentynineteen/
Author: the WordPress team
Author URI: https://wordpress.org/
Description: Our 2019 default theme is designed to show off the power of the
block editor. It features custom styles for all the default blocks, and is
built so that what you see in the editor looks like what you'll see on your
website. Twenty Nineteen is designed to be adaptable to a wide range of
websites, whether you're running a photo blog, launching a new business, or
supporting a non-profit. Featuring ample whitespace and modern sans-serif
headlines paired with classic serif body text, it's built to be beautiful on
all screen sizes.
Requires at least: WordPress 4.9.6
Version: 1.4
License: GNU General Public License v2 or later
License URI: LICENSE
Text Domain: twentynineteen
Tags: one-column, flexible-header, accessibility-ready,
custom-colors, custom-menu, custom-logo, editor-style,
featured-images, footer-widgets, rtl-language-support,
sticky-post, threaded-comments, translation-ready
This theme, like WordPress, is licensed under the GPL.
Use it to make something cool, have fun, and share what
you've learned with others.
Twenty Nineteen is based on Underscores https://underscores.me/,
(C) 2012-2018 Automattic, Inc.
Underscores is distributed under the terms of the GNU GPL v2 or later.
Normalizing styles have been helped along thanks to the fine work of
Nicolas Gallagher and Jonathan Neal https://nicolas.github.io/normalize.css/
*/
```

Файл `style.css` активной темы (и родительской темы, если допустимо) автоматически применяется WordPress.

Создание версий CSS-файлов вашей темы

Хорошая практика — установка версии для ваших CSS-файлов при загрузке их через `wp_enqueue_style()`. Таким образом, если вы обновите свой CSS, то сможете обновить номер версии и не допустить, чтобы пользователи сайта видели на первый взгляд неработающий сайт, который использует кэшированную браузером версию таблицы стилей.

Когда WordPress применяет файл `style.css` вашей темы, он загружает таблицу стилей с учетом полной версии WordPress. Вывод строки в заголовке вашего сайта будет выглядеть так:

```
<link rel='stylesheet'
      id='twentynineteen-style-css'
      href='../wp-content/themes/twentynineteen/style.css?ver=1.4'
      type='text/css'
      media='all' />
```

Обновления таблицы стилей, номера версии вашего приложения или даже номера версии, заданного в комментарии `style.css`, не обновят версию, добавленную в таблицу стилей при применении. Она всегда будет соответствовать номеру версии WordPress.

Одно из возможных решений — удаление всего кода CSS из вашего файла `style.css` в другие файлы CSS вашей темы и загрузка *этих файлов* CSS через вызовы функции `wp_enqueue_style()` в файле `functions.php` темы. Следующий пример иллюстрирует, как это будет выглядеть для `style.css`.

Пример

```
/*
Theme Name: SchoolPress
Version: 1.0

Вот и все! Весь CSS можно найти в папке "css" темы.
*/

// И вот что будет в functions.php:

<?php
define('SCHOOLPRESS_VERSION', '1.0');
function sp_enqueue_theme_styles() {
    if (!is_admin()) {
        wp_enqueue_style('schoolpress-theme',
            get_stylesheet_directory_uri() . '/css/main.css',
```

```

        NULL,
        SCHOOLPRESS_VERSION
    );
}

add_action('init', 'sp_enqueue_theme_styles');
?>

```

Константа, такая как `SCHOOLPRESS_VERSION`, обычно определяется в нашем основном файле плагина, но она включена здесь для ясности. Предыдущий код загружает наш новый файл `/css/main.css` с добавленной основной версией приложения, поэтому новые версии приложения не будут конфликтовать с таблицами стилей, кэшированными браузером.

Существует другой способ изменить версию основного файла `style.css`, не перемещая его в другой файл полностью — фильтр `wp_default_styles`, который передает объект, содержащий значения по умолчанию, используемые при загрузке таблицы стилей. Одним из этих значений является `default_version`, которое можно изменить следующим образом:

```

define('SCHOOLPRESS_VERSION', '1.0');
function sp_wp_default_styles($styles)
{
    // использовать версию релиза для таблиц стилей
    $styles->default_version = SCHOOLPRESS_VERSION;
}
add_action("wp_default_styles", "sp_wp_default_styles");

```

Теперь наша основная таблица стилей будет загружена согласно версии приложения SchoolPress вместо основной версии WordPress. Мы можем сохранить наш CSS-код в файле `style.css`, если захотим, однако целесообразнее переместить хотя бы некоторые части CSS в отдельные файлы в папке `/css` вашей темы.

Файл `functions.php`

Файл `functions.php` вашей активной темы (и родительской темы, если применимо) загружается каждый раз при запуске WordPress. Поэтому файл `functions.php` является популярным местом для добавления небольших программистских трюков и других случайных фрагментов кода. На типичном сайте WordPress файл `functions.php` может быстро потерять упорядоченность.

Однако мы разрабатываем хорошо спланированное приложение WordPress, и *наши файлы* `function.php` не должны быть беспорядочными. Аналогично тому, как мы разбиваем функции нашего основного плагина приложения на более мелкие файлы, вы должны сделать то же самое с файлом `functions.php` вашей темы. В папку вашей темы вы можете добавить файлы, подобные следующим:

- ♦ `/includes/functions.php` — для размещения вспомогательных функций;

- ♦ `/include/settings.php` — для кода, связанного с настройками и параметрами темы;
- ♦ `/include/sidebar.php` — чтобы определить боковые панели/области виджетов.

Кроме того, убедитесь, что программный код, который вы добавляете в файл `functions.php` вашей темы, связан с внешним интерфейсом вашего сайта. Код, который применяется к панели администрирования WordPress, серверной обработке вашего приложения или всего приложения в целом, скорее всего, должен быть где-то в основном плагине приложения.

Темы и CPT

Так как CPT — это просто записи, по умолчанию ваши CPT будут отображаться с использованием шаблона `single.php` или `index.php`, если не доступен шаблон `single.php`. Вы также можете добавить файл к своей теме в виде `<post_type>.php`, где `<post_type>` — это слаг вашей CPT. Если этот файл доступен, он служит для визуализации представления одного сообщения этого типа.

Точно так же вы можете добавить файл `archive-<post_type>.php` для отображения архива вашего CPT, если в определении CPT установлен флаг `has_archive`. Мы рассмотрим CPT (включая указание шаблонов для них) более подробно в *главе 5*.

Популярные фреймворки для разработки тем

При создании приложений с помощью WordPress вам доступно множество фреймворков для тем, как специфичных для WordPress, так и общего назначения HTML/CSS. Независимо от того, собираетесь ли вы использовать фреймворк для быстрой проверки концепции или применить его в качестве основного компонента вашей пользовательской темы, фреймворк темы поможет сэкономить вам много времени.

Мы кратко рассмотрим некоторые распространенные фреймворки тем и уделим особое внимание двум самым популярным из них для разработки приложений WordPress.

Но сначала выясним, что представляет собой фреймворк при разработке темы.

Фреймворки тем WordPress

Фреймворки для тем WordPress — это темы, которые предназначены для использования в качестве родительских тем или начальных тем для ускорения разработки веб-интерфейса. Типичные фреймворки для разработки тем, включают в себя основные стили и макеты для сообщений блога, а также архивы, страницы, боковые панели и меню. Они имеют различный объем, а некоторые включают классы CSS, шорткоды и другие полезные фрагменты кода, которые помогут вам создавать но-

вые макеты и добавлять элементы пользовательского интерфейса на свои страницы. Все фреймворки, скорее всего, сэкономят вам много времени.

Есть две причины предпочесть одну платформу тем перед другой: либо вы выберете дочернюю тему, которая визуально выглядит очень близко к вашему видению вашего приложения, либо выбираете фреймворк, который написан так, что вам удобно работать с ним.

_s (underscores)

Исходная тема `_s` (произносится как "underscores") публикуется Automattic и содержит все общие компоненты, необходимые для темы WordPress. В отличие от большинства других фреймворков, тема `_s` предназначена не для использования в качестве родительской темы, а в качестве отправной точки для разработки вашей собственной темы. Большинство тем, разработанных Automattic для WordPress.com, основаны на теме `_s`.

Для работы с фреймворком `_s`, скачайте его код и измените имя каталога и все ссылки на `_s` на название вашей темы. Неплохие инструкции, как это сделать, можно найти в файле проекта *README* (bwwawp.com/s-readme) или, что еще лучше, это можно сделать в автоматическом режиме — для вас на сайте [underscores](http://underscores.me) есть автоматический инструмент (www.underscores.me).

Таблица стилей в `_s` очень минималистична, без реальных стилей, просто немного кода для разметки и некоторые общие настройки для удобства. Тема `_s` лучше всего подходит для дизайнеров, которые могут и хотят создать свою собственную тему с нуля. Это в основном код, который вам придется написать для своей темы самостоятельно. Код темы `_s` не настолько абстрагирован, как некоторые другие тематические фреймворки, поэтому применение фреймворка должно быть проще для разработчиков, более знакомых с HTML и CSS, чем с PHP.

Memberlite

Memberlite — это тема, написанная Джейсоном Коулманом и Кимберли Коулманом из Stranger Studios. Она была создана в основном для сайтов сообществ, но может использоваться на самых разных сайтах. Слово "Lite" в заголовке намекает на облегченный дизайн темы (в техническом и эстетическом плане).

Memberlite разработана так, чтобы уместиться на экранах разных размеров. Здесь есть шаблоны и разделы для всего, что нужно современному сайту. Она имеет несколько сопутствующих плагинов (Memberlite Elements и Memberlite Shortcodes), с инструментами для более точного управления боковыми панелями, баннерами, макетами страниц и другими компонентами вашего сайта.

Memberlite лучше всего подходит для дизайнеров-разработчиков и действительно является нашим выбором, который основан на балансе поддержки фреймворка как дизайна, так и кода.

Genesis

Genesis (bawawp.com/genesis) — это фреймворк тем, разработанный StudioPress и используемый в более чем 40 дочерних темах, опубликованных StudioPress, и во многих других темах, опубликованных сторонними дизайнерами. Тема Genesis предназначена для применения в качестве родительской темы. StudioPress имеет дочерние темы, которые подходят для различных типов бизнеса и веб-сайтов. Можно создать и свою собственную дочернюю тему, которая наследуется от Genesis.

Каркас Genesis абстрагирует базовый HTML и CSS больше, чем другие фреймворки, перечисленные здесь. Мы находим, что с ним немного сложнее работать при выполнении масштабных настроек. Тем не менее Genesis будет хорошим выбором, если вы обнаружите, что одна из его дочерних тем подходит вам на 80% или если вам кажется, что с его структурой проще работать, чем с другими вариантами.

Сторонние фреймворки тем

В дополнение к темам WordPress существуют также фреймворки для разработки пользовательского интерфейса приложения, которые предоставляют разметку, таблицы стилей и изображения для общих шаблонов пользовательского интерфейса и элементов. Некоторые популярные пользовательские интерфейсы включают Bootstrap Twitter (getbootstrap.com) и Foundation Zurb (foundation.zurb.com).

Встраивание фреймворка пользовательского интерфейса в вашу тему может быть таким же простым, как копирование нескольких файлов в папку темы и применение таблиц стилей и JavaScript. Это дает вам легкий доступ к стилизованным элементам пользовательского интерфейса, таким как кнопки, вкладки, нумерация страниц, области навигации, ярлыки, оповещения и индикаторы выполнения.

Далее мы рассмотрим, как добавить ресурсы Bootstrap в дочернюю тему Memberlite, но тот же процесс должен работать для других комбинаций тем WordPress и фреймворков пользовательского интерфейса.

Создание дочерней темы для Memberlite

Чтобы создать свою тему, вам необходимо выполнить следующие шаги:

1. Создайте новую папку в папке wp-content/themes.
Затем задайте ей имя memberlite-child.
2. Создайте файл style.css в папке memberlite-child.
3. Вставьте код следующего примера в файл style.css.

Пример

```
/*
THEME NAME: Memberlite Child
THEME URI: http://bawawp.com/wp-content/themes/memberlite-child/
DESCRIPTION: Memberlite Child Theme
```

```
VERSION: 0.1
AUTHOR: Jason Coleman
AUTHOR Uri: http://bwawwp.com
TAGS: memberlite, child, tag
TEMPLATE: memberlite
*/
@import url("../memberlite/style.css");
```

Ключевое поле в комментарии — `TEMPLATE`, которое должно соответствовать папке родительской темы, в данном случае `memberlite`. Единственный необходимый файл для дочерней темы — `style.css`. Итак, на данный момент вы создали дочернюю тему.

Вы можете либо скопировать весь код CSS из `style.css` родительской темы в `style.css` дочерней темы и отредактировать то, что хотите, либо выполнить директиву `@import url`, как мы это сделали здесь, чтобы импортировать правила из таблицы стилей родительской темы, и добавить дополнительные правила ниже, чтобы переопределить стили родительской темы.

Чтобы применить файлы при начальной загрузке, вам также потребуется файл `functions.php`.

Создайте пустой файл `functions.php` в папке `memberlite-child`.

Включение Bootstrap в тему вашего приложения

В общем, импорт Bootstrap в тему Memberlite можно сравнить с простым поиском темы на основе Bootstrap или просто копированием нужных вам правил CSS. Однако вы можете столкнуться с импортом фреймворков и библиотек в свою тему. Следующая инструкция даст вам представление о том, как импортировать другие библиотеки и фреймворки в вашу тему.

Скачайте ZIP-файл Bootstrap (getbootstrap.com) в папку `memberlite-child`. После распаковки у вас будет папка `dist`, содержащая файлы CSS и JavaScript для Bootstrap. Вы можете переименовать эту папку в `bootstrap` и удалить ZIP-файл Bootstrap. Папка вашей дочерней темы теперь должна выглядеть так:

```
memberlite-child
bootstrap
/css/
js
functions.php
style.css
```

Теперь мы применим Bootstrap CSS и JavaScript, добавив приведенный далее код примера в файл `functions.php` внутри вашей дочерней темы.


```

<?php
function memberlite_child_init() {
    wp_enqueue_style(
        'bootstrap',
        get_stylesheet_directory_uri() .
        '/bootstrap/css/bootstrap.min.css',
        'style',
        '3.0'
    );
    wp_enqueue_script(
        'bootstrap',
        get_stylesheet_directory_uri() .
        '/bootstrap/js/bootstrap.min.js',
        'jquery',
        '3.0'
    );
}
add_action('init', 'memberlite_child_init');
?>

```

Обратите внимание, что мы устанавливаем зависимости для стиля Bootstrap CSS, чтобы обеспечить загрузку таблицы стилей Bootstrap после таблицы стилей Memberlite. Мы также настроили зависимость JavaScript Bootstrap от jquery и установили версию обоих файлов на 3.0, соответствующую версии Bootstrap.

На данный момент вы можете выбрать любой из ваших любимых стилей Bootstrap или JavaScript в вашей теме WordPress. Многие стили Bootstrap для столбцов и разметки отсутствуют в разметке Memberlite (Memberlite имеет собственную систему макетов), поэтому они не будут применяться к вашей теме. Но стили для элементов формы и кнопок были бы полезны для разработчиков приложений.

Меню

Меню являются важной частью большинства приложений, и приложения часто предъявляют особые требования к меню, которых нет у других веб-сайтов. Некоторые приложения имеют несколько меню. Многие мобильные приложения включают главное навигационное меню вверху и меню, похожее на панель инструментов внизу страницы. Некоторые приложения снабжены динамическими меню. Меню во многих приложениях различается для вошедших в систему пользователей и для не вошедших в систему пользователей. Пункты меню могут быть основаны на уровне членства пользователя или его правах.

Прежде чем мы начнем создавать более сложные меню и элементы навигации с помощью WordPress, давайте рассмотрим стандартный способ добавления меню в вашу тему.

Навигационные меню

Начиная с версии WordPress 3.0, стандартный метод добавления меню навигации к темам включал в себя регистрацию меню в коде темы, указание места, в котором будет отображаться меню, а затем управление этим меню с помощью панели управления WordPress.

Основное преимущество встроенных функций меню WordPress заключается в том, что конечные пользователи могут управлять содержимым своих меню с помощью графического интерфейса панели управления. Даже если вы являетесь разработчиком, обладающим полным контролем над своим приложением, все же лучше использовать встроенные меню в WordPress, поскольку у вас могут быть заинтересованные лица, которые захотят управлять меню, или вы можете распространять свою тему среди других пользователей в будущем. В WordPress меню навигации также очень легко перемещать, и они могут реализовать преимущества другого кода с помощью связанных с меню хуков или стилей CSS.

Зарегистрировать новое навигационное меню можно с помощью функции `register_nav_menu($location, $description)`. Параметр `$location` — это уникальный фрагмент, предназначенный для идентификации меню. Параметр `$description` — это более длинный заголовок для меню, показанного в раскрывающемся списке в нашем инструменте для управления меню на панели инструментов:

```
register_nav_menu('main', 'Main Menu');
```

Функция `register_nav_menus()` (с `s` на конце) позволяет зарегистрировать много меню одновременно. Она принимает массив местоположений, в которых ключи являются слагами `$location`, а значения — заголовками `$description`:

```
register_nav_menus(array(
    'main' => 'Main',
    'logged-in' => 'Logged-In'
));
```

Функция `wp_nav_menu()` дает возможность разместить навигационное меню в своей теме:

```
wp_nav_menu(array('theme_location' => 'main'));
```

Параметр `theme_location` должен быть установлен в значение `$location`, заданный с помощью функции `register_nav_menu()`. Функция `wp_nav_menu()` может принимать множество других параметров, чтобы изменить поведение и разметку меню. Страница Кодекса WordPress для меню навигации bwwwp.com/nav-codex — хороший ресурс, где собраны различные параметры для функции `wp_nav_menu()` и другие способов настроить меню. Мы рассмотрим некоторые из наших любимых рецептов в следующих разделах.

Динамические меню

Существуют два основных способа сделать ваши меню WordPress динамичными, чтобы на разных страницах или при различных условиях отображались разные пункты меню. Первый — настроить два меню и загрузить одно или другое меню в зависимости от ситуации. Вот фрагмент кода из Кодекса, показывающий, как отобразить разные меню для вошедших и вышедших из приложения пользователей:

```
if (is_user_logged_in()) {
    wp_nav_menu(array('theme_location' => 'logged-in-menu'));
} else {
    wp_nav_menu(array('theme_location' => 'logged-out-menu'));
}
```

Другой способ сделать ваше меню динамичным — использовать фильтр `nav_menu_css_class` для добавления дополнительных классов CSS к определенным пунктам меню. Затем с помощью CSS вы можете скрыть/показать определенные пункты меню в зависимости от их класса CSS.

Скажем, вы хотите удалить ссылку входа в меню, когда вы находитесь на странице входа³. Приведенный далее пример кода иллюстрирует, как это можно сделать.

Пример

```
function remove_login_link($classes, $item)
{
    if(is_page('login') && $item->title == 'Login')
        $classes[] = 'hide';      // скрыть этот пункт

    return $classes;
}

add_filter('nav_menu_css_class', 'sp_nav_menu_css_class', 10, 2);
```

Вы также можете настроить разметку своих меню с помощью пользовательских классов Walker (см. главу 7).

Адаптивный дизайн

Мы могли бы написать еще одну книгу об адаптивном дизайне. К счастью для нас, многие уже сделали это, например Кларисса Петерсон, — автор *Learning Responsive Web Design (oreil.ly/learn-rwd)*. Общая концепция адаптивного дизайна заключается в том, чтобы правильно определить свойства клиентского устройства и настроить макет, дизайн и функциональные возможности вашего приложения для оп-

³ Вы можете проверить `$ _SERVER ['PHP_SELF']`, чтобы увидеть, находитесь ли вы на странице `wp-login.php`. В этом примере мы предполагаем, что наша страница входа находится на странице WordPress со слагом `login`.

тимальной работы на этом устройстве. Теперь давайте рассмотрим несколько различных способов сделать это.

Определение устройства и дисплея с помощью CSS

Медиазапросы — основной метод определения устройств в CSS. Они присутствуют в таблицах стилей или добавляются как свойство тега `<link>` для встраивания таблицы стилей, чтобы ограничить область применения правил CSS конкретным типом носителя или случаем, в котором доступна особая функция медиа. Mozilla (mzl.la/css-mq) хорошо объясняет медиазапросы и перечисляет различные свойства и операторы, которые пригодны для построения медиазапроса.

Распространенное использование медиазапросов — скрывание определенных элементов и настройка шрифта и размеров элементов, когда кто-то печатает. Самый простой пример — указать медиазапрос в теге `<link>`, внутри таблицы стилей или вызвать функцию `wp_enqueue_style`, как в следующем фрагменте кода.

Пример

```
<link rel="stylesheet" media="print" href="example.css" />

<style>
@media print
{
    .hide-from-print {display: none;}
    .show-when-printing {display: auto;}
}
</style>

<?php
    wp_enqueue_style('example', 'example.css', NULL, '1.0', 'print');
?>
```

Более типичный пример в мире адаптивного дизайна — проверка минимальной и/или максимальной ширины в медиазапросе для корректировки стилей по мере уменьшения или увеличения размера экрана. Далее приведен пример из таблицы стилей Bootstrap, которая настраивает правила CSS для экранов в диапазоне от 768 до 979 пикселей (текущая ширина типичного окна браузера на мониторе). Размеры, превышающие 979 пикселей, можно считать очень большими.

Пример

```
@media (min-width: 768px) and (max-width: 979px) {
    .hidden-desktop {
        display: inherit !important;
    }
}
```

```

.visible-desktop {
    display: none !important ;
}
.visible-tablet {
    display: inherit !important;
}
.hidden-tablet {
    display: none !important;
}
}

```

Другая распространенная задача, решаемая с помощью медиазапросов, — это изменение стилей и, в частности, замена изображений, когда в браузере установлен экран Retina с высоким разрешением⁴.

Вот совокупность медиазапросов, используемых в некоторых CSS-панелях WordPress для обнаружения дисплея с высоким разрешением. Запросы проверяют соотношение пикселей и DPI. Значения различаются для разных дисплеев, но большинство дисплеев со стандартным разрешением имеют соотношение пикселей 1:1 и 96 DPI. Соотношение пикселей для дисплея Retina 2:1 и DPI 196 или выше, но мы можем проверить минимальные значения где-то между стандартным разрешением и определением уровня Retina, чтобы не упустить другие дисплеи высокого разрешения:

```

@media(-o-min-device-pixel-ratio: 5/4),           /* Opera */
      (-webkit-min-device-pixel-ratio: 1.25),     /* Webkit */
      (min-resolution: 120dpi) {                  /* Прочие */
    / * добавьте свой CSS высокого разрешения здесь * /
}

```

Медиазапросы — мощный инструмент, и вы на их основе можете создавать очень гибкие пользовательские интерфейсы. Браузеры и стандарты CSS постоянно развиваются. Важно быть в курсе событий, чтобы современные телефоны, планшеты и мониторы отображали ваше приложение так, как вы хотите.

Вопрос, какие свойства нужно искать и как настроить таблицу стилей для их соответствия, выходит за рамки этой книги, но, надеюсь, вы поймете идею и сумеете включить медиазапросы в ваши темы WordPress.

Определение устройств и их свойств в JavaScript

С помощью JavaScript также можно определить свойства и функции устройства пользователя вашего приложения. jQuery предлагает методы определения разме-

⁴ Retina — это торговая марка Apple, известная своими дисплеями с высоким разрешением. Однако вы можете найти термин "retina" в комментариях к коду программ и документации для обозначения любого дисплея с высоким разрешением.

ров окна и экрана и выдает другую информацию о браузере. Многие функции HTML5, по-разному доступные в том или ином браузере, могут быть протестированы перед использованием.

Определение размера экрана и окна с помощью JavaScript и jQuery

В JavaScript ширина и высота экрана доступны в свойствах `screen.width` и `screen.height`. Свойства `screen.availWidth` и `screen.availHeight`, тоже позволяют получить доступную ширину и высоту с учетом пикселей, занятых панелями инструментов и боковыми панелями в окне браузера.

Если вы уже пользуетесь jQuery, вы можете задействовать метод `width()` для любого элемента на вашей странице, чтобы получить его ширину, и вы также можете использовать его для объектов `$(document)` и `$(window)`, чтобы получить ширину документа и окна соответственно. А еще есть свойство `height()` для объектов документа и окна или любого элемента на вашей странице.

Значения для `$(window).width()` и `$(window).height()` должны совпадать со значениями `screen.availWidth` и `screen.availHeight`. Это доступный размер окна просмотра браузера, за вычетом любых панелей инструментов или боковых панелей или, точнее, сколько у вас есть места для отображения HTML.

Ширина и высота объектов `$(document)` будет соответствовать общей ширине прокручиваемой страницы и высоте отображаемой веб-страницы. При использовании ширины и высоты в коде JavaScript, вы часто захотите обновить данные, если размер окна изменится. Это может произойти, если кто-то поменяет размер окна браузера на рабочем столе, повернет телефон из портретной в альбомную ориентацию или выполнит любое из действий, которые могут изменить ширину или высоту окна. jQuery предлагает простой способ обнаружить эти изменения, чтобы вы могли соответствующим образом обновить свой макет:

```
// привязываем событие, чтобы запускать этот код при изменении размера окна
jQuery(window).resize(function() {
    width = jQuery(window).width();
    height = jQuery(window).height();
    // обновить макет и т. д
});
```

Вы можете привязать событие изменения размера к любому элементу, а не только к полному окну. Элементы на вашей странице могут расти и сжиматься, когда пользователь взаимодействует с вашей страницей, возможно добавляя элементы через формы Ajax, перетаскивая изменяемые размеры элементов на экране или иным образом перемещая объекты.

Обнаружение функций в JavaScript

Когда вы создаете современный пользовательский интерфейс приложения с использованием функций HTML5, иногда вам может понадобиться определить, доступна ли

определенная функция HTML5, чтобы предоставить альтернативу или запасной вариант. Книга "Погружение в HTML5" Марка Пилигрима (diveintohtml5.info) содержит хороший перечень общих методов обнаружения функций HTML5:

- ◆ Проверьте, существует ли определенное свойство в глобальном объекте (таким как `window` или `navigator`).
- ◆ Создайте элемент, затем проверьте, существует ли определенное свойство для этого элемента.
- ◆ Создайте элемент, выясните, существует ли определенный метод для этого элемента, а затем вызовите метод и проверьте значение, которое он возвращает.
- ◆ Создайте элемент, установите для свойства определенное значение, а затем проверьте, сохранило ли свойство свое значение.

Если вам нужно выполнить только одну такую проверку, то рекомендуем ознакомиться с примерами на сайте *Dive into HTML5*. Если вам требуется много и часто обнаруживать подобные функции, то здесь поможет библиотека типа `Modernizr.js` (modernizr.com).

Чтобы воспользоваться `Modernizr.js`, сначала скачайте нужную версию скрипта с их сайта. `Modernizr` предлагает инструмент, который спросит вас, какие части скрипта вам нужны, а затем создаст небольшой файл `*.js`, содержащий только эти фрагменты. Поместите этот файл в папку с темой или плагином и активируйте его, как показано в следующем примере кода.

Пример

```
<?php
function sp_wp_footer_modernizr() {
    wp_enqueue_script(
        'modernizr',
        get_stylesheet_directory_uri() . '/js/modernizr.min.js'
    );?>
    <script>
        // изменить входные данные поиска на текст, если не поддерживается
        if(!Modernizr.inputtypes.search)
            jQuery('input[type=search]').attr('type', 'text');
    </script>
<?php
}
add_action('wp_footer', 'sp_wp_footer_modernizr');
?>
```

Документация `Modernizr` (bawawp.com/modern-doc) содержит список функций, обнаруживаемых с помощью `Modernizr.js`.

`jQuery` также предоставляет аналогичный набор проверок, ограниченный ситуациями, которые `jQuery` должен проверять через объект `jQuery.support`. Если провер-

ка, которую вы пытаетесь сделать, уже выполнена jQuery, вы можете избежать накладных расходов Modernizr.js, реализовав проверку jQuery. Список флагов функций, установленных jQuery.support, можно найти на веб-сайте (bwwwp.com/jquery-doc):

```
jQuery(document).ready(function() {  
    // загружать код AJAX только если AJAX доступен  
    if(jQuery.support.ajax)  
    {  
        // AJAX код будет здесь  
    }  
});
```

Определение устройства в PHP

Определение устройства с помощью PHP основано на глобальной переменной `$_SERVER['HTTP_USER_AGENT']`, созданной PHP. Это значение устанавливается самим браузером и поэтому определено не является стандартным. Оно часто вводит в заблуждение и потенциально подделывается веб-сканерами и другими ботами. Лучше всего избегать определения браузера устройства на основе PHP, делая свой код как можно более стандартизированным и использовать методы CSS и JavaScript для определения функций.

Если вы хотите получить общее представление о том, какой браузер обращается к вашему приложению, лучше всего указать строку User agent. В следующем фрагменте кода приведен простой тестовый скрипт, отображающий эту строку, и пример того, как это будет выглядеть.

Пример

```
<?php  
echo $_SERVER['HTTP_USER_AGENT'];  
  
/*  
    Выводит что-то вроде:  
    Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4)  
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.1500.95 Safari/537.36  
*/  
>
```

Эта строка агента пользователя содержит полезную информацию, но, возможно, в ней слишком много сведений. В строке указано не менее пяти разных имен браузеров. Так что это за браузер? Mozilla, KHTML, Gecko, Chrome или Safari? В данном случае мы работали с Chrome на MacBook Air под управлением OS X.

Мы уже упоминали, что не существует стандарта для пользовательских агентов, которые будет отправлять браузер. Исторически, браузеры включают имена ста-

рых браузеров, которые в основном говорят: "Я могу делать все, что делает этот браузер".

На WebAIM (oreil.ly/7i7PZ) есть забавная история различных строк пользовательских агентов, включая эту часть, объясняющую "родословную" браузера Chrome:

"А затем Google создал Chrome, а Chrome использовал Webkit, и он был похож на Safari, и хотел, чтобы страницы создавались для Safari, и поэтому притворялся Safari. Таким образом, Chrome использовал WebKit и претендовал на звание Safari, а WebKit — на KHTML, а KHTML — на Gecko, а все браузеры — на Mozilla, а Chrome называл себя Mozilla/5.0 (Windows; U; Windows NT 5.1; ru-ru) AppleWebKit / 525.13 (KHTML, как Gecko) Chrome/0.2.149.27 Safari/525.13, и строка агента пользователя была почти бесполезной, и все притворялись, и была полная путаница".

Аарон Андерсон

Обнаружение браузера в ядре WordPress

К счастью, WordPress проделал небольшую работу по синтаксическому анализу строки пользовательского браузера и предоставляет некоторые глобальные переменные и несколько методов, которые охватывают большинство общих вопросов, связанных с определением браузера. Следующие глобальные переменные установлены WordPress в `wp-includes/vars.php`:

```
$is_lynx, $is_gecko, $is_winIE, $is_macIE, $is_opera, $is_NS4, $is_safari, $is_chrome, $is_iphone, $is_IE.
```

И для обнаружения определенных серверов у нас есть следующее:

```
$is_apache, $is_IIS, $is_iis7.
```

Наконец, вы можете использовать функцию `wp_is_mobile()`, которая проверяет на наличие слова *mobile* в строке агента пользователя, а также несколько распространенных мобильных браузеров.

Далее приведен простой пример, показывающий, как вы можете с помощью глобальных переменных загружать различные скрипты и CSS.

Пример

```
<?php
function sp_init_browser_hacks() {
    global $is_IE;
    if ($is_IE) {
        // проверяем версию и загружаем CSS
        $user_agent = strtolower($_SERVER['HTTP_USER_AGENT']);
        if (strpos('msie 6.', $user_agent) !== false &&
            strpos('opera', $user_agent) === false) {
            wp_enqueue_style(
                'ie6-hacks',
```

```

        get_stylesheet_directory_uri() . '/css/ie6.css'
    );
}

}

if (wp_is_mobile()) {
    // загрузить наш CSS и JS для мобильных устройств
    wp_enqueue_style(
        'sp-mobile',
        get_stylesheet_directory_uri() . '/css/mobile.css'
    );
    wp_enqueue_script(
        'sp-mobile',
        get_stylesheet_directory_uri() . '/js/mobile.js'
    );
}

}

add_action('init', 'sp_init_browser_hacks');
?>

```

Обнаружение браузера с помощью PHP-функции `get_browser()`

PHP на самом деле имеет отличную встроенную функцию обнаружения браузера: `get_browser()`. Далее приведен простой пример вызова `get_browser()` и отображения некоторых типичных результатов.

Пример

```

<?php
$browser = get_browser();
print_r($browser);

```

/*

Вывод будет примерно таким:

```

stdClass Object (
    [browser_name_regex] => $^mozilla/5\.0 \((.*intel mac os x.*\)
applewebkit/. * \ (khtml, like gecko\).*chrome/28\..*safari/. *$$
    [browser_name_pattern] => Mozilla/5.0 (*Intel Mac OS X*)
AppleWebKit/* (KHTML, like Gecko)*Chrome/28.*Safari/*
    [parent] => Chrome 28.0
    [platform] => MacOSX
    [win32] =>
    [comment] => Chrome 28.0
    [browser] => Chrome

```

```

[version] => 28.0
[majorver] => 28
[minorver] => 0
[frames] => 1
[iframes] => 1
[tables] => 1
[cookies] => 1
[javascript] => 1
[javaapplets] => 1
[cssversion] => 3
[platform_version] => unknown
[alpha] =>
[beta] =>
[win16] =>
[win64] =>
[backgroundsounds] =>
[vbscript] =>
[activexcontrols] =>
[ismobiledevice] =>
[issyndicationreader] =>
[crawler] =>
[aolversion] => 0
)
*/

```

Результат не может не удивить! Так почему же эта функция упоминается последней в разделе об определении браузера с помощью PHP? Ответ заключается в том, что функция `get_browser()` недоступна или устарела на большинстве серверов. Чтобы функция предоставляла вам полезную информацию или в большинстве случаев работала вообще, вам необходимо загрузить обновленный файл `browscap.ini` и настроить PHP, чтобы найти его. Если вы распространяете свое приложение, то придется выбрать другой метод для определения возможностей браузера. Однако если вы запускаете свое собственное приложение на своих собственных серверах, то `get_browser()` является честной игрой.

Актуальный файл `browscap.ini` можно найти на веб-сайте проекта Browser Capabilities (browscap.org). Убедитесь, что вы получили один из файлов, отформатированных для PHP. Мы рекомендуем файл `lite_php_browscap.ini`, который в два раза меньше, но содержит информацию о самых популярных браузерах.

После того, как на вашем сервере появится файл `*.ini`, вам нужно обновить файл `php.ini` и указать на новый файл. Ваш файл `php.ini`, вероятно, содержит закомментированную строку `browscap`. Раскомментируйте ее и убедитесь, что она указывает на местоположение файла `*.ini`, который вы скачиваете. Все должно выглядеть примерно так:

```

[browscap]
browscap = /etc/lite_php_browscap.ini

```

Теперь перезапустите ваш веб-сервер (Apache, Nginx и т. д.), и функция `get_browser()` должна работать.

Последнее замечание по определению браузера

Мы потратили много времени на определение браузера, но на практике вам следует использовать его только в качестве крайней меры. Когда у вас возникают неприятности с определенным браузером из-за особенностей дизайна или функциональности, появляется соблазн попытаться обнаружить его и выстроить процесс программирования вокруг него. Однако, если возможно найти другой обходной путь, который дает аналогичный результат без выделения конкретных браузеров, обычно такое решение предпочтительнее.

Во-первых, как мы видели здесь, строка пользовательского агента не имеет стандартов, и вам может потребоваться регулярно обновлять код, чтобы проанализировать его с учетом новых браузеров и версий браузеров.

Во-вторых, в некоторых случаях проблема, связанная с браузером, является признаком более серьезной проблемы в вашем коде. Может быть, следует поискать способ упростить ваш дизайн или функциональность для лучшей работы в разных браузерах, устройствах и экранах разных размеров.

Конечная цель адаптивного дизайна и программирования — создать нечто, что будет достаточно гибким для учета всех браузеров и клиентов, обращающихся к вашему приложению, независимо от того, знаете ли вы о них или нет.

Пользовательские типы записей, метаданные записей и таксономия

СРТ — это то, что действительно делает WordPress системой управления контентом. С их помощью вы можете быстро создавать пользовательские функции и хранить данные в согласованном порядке.

Типы сообщений по умолчанию и СРТ

При установке WordPress по умолчанию у вас уже есть несколько типов записей. Типы записей, которые вам наиболее знакомы, это страницы и посты, но есть еще несколько. Все эти значения типов записей хранятся в таблице базы данных `wp_posts` и подразделяются согласно данным в поле `post_type`.

Страница

Страницы WordPress — это статические страницы контента, такие как домашняя страница, страница "о сайте", контактная информация, биографии или любая другая пользовательская страница, которую вы хотите создать. Страницы могут быть неограниченно вложены в любую иерархическую структуру. Они также могут быть отсортированы по значению поля `menu_order`.

Публикация

Ваши публикации — это ваш блог или новости, или называйте как хотите, сюда входит весь ваш контент, который индексируется поисковыми системами Интернета. Вы можете классифицировать свои посты, помечать их ключевыми словами, устанавливать даты публикации и многое другое. В общем, публикации отображаются в виде списка в обратном хронологическом порядке на веб-интерфейсе вашего сайта.

Вложение

Всякий раз, когда вы загружаете изображение или файл в публикацию, WordPress сохраняет этот файл не только на сервере, но и как публикацию в таблице `wp_posts` с значением `attachment` в поле `post_type`.

Редакции

WordPress заботится о вас и сохраняет ваши посты в качестве редакций каждый раз, когда вы или кто-либо другой редактирует их. Эта функция включена по умолчанию, и вы можете обратиться к истории, чтобы вернуть содержимое к предыдущему состоянию, если что-то напутано.

Иногда, если ваше приложение настроено на внесение множества изменений в поле `post_content`, таблица `wp_posts` заполняется после всех изменений. Поэтому вы можете захотеть ограничить количество редакций, хранящихся в таблице `wp_posts`. Для этого добавьте следующий код в файл `wp-config.php`:

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

Здесь 5 — это количество редакций, которые нужно сохранить для данного поста. Значение 0 отключит публикацию предыдущих версий. Значение `true` или `-1` будет хранить бесконечное количество версий (для этого может потребоваться много места на диске).

Элемент меню навигации

Каждый раз, когда вы создаете пользовательское меню с помощью основного редактора меню WordPress (**wp-admin** → **appearance** → **menus**), вы храните записи с информацией для ваших меню.

Пользовательский CSS

Объект Customizer имеет настройку "Дополнительный CSS". Этот код CSS хранится в настраиваемой записи типа `custom_css`. Дополнительный CSS специфичен для активной темы, и у каждой темы будет отдельная запись `custom_css` для хранения CSS. Может показаться странным хранить эти дополнительные правила CSS в пользовательском типе записи, но это было сделано для удобства представления. Вы можете представить себе сайт с *множеством* пользовательских настроек CSS. Параметры иногда автоматически загружаются или кэшируются, и эти системы могут замедлиться или даже "зависнуть" при большом количестве CSS.

Наборы изменений

Наборы изменений похожи на версии публикации, но для ваших настроек, а не для постов. При редактировании настроек в настройщике эти изменения сохраняются в записи типа `customize_changeset`. Если вы случайно закрыли вкладку браузера и вернулись в настройщик, то WordPress предложит вам восстановить эти изменения с помощью набора изменений.

Кэш oEmbed

WordPress позволяет встраивать контент от поддерживаемых провайдеров oEmbed, помещая URL-адрес контента в отдельной строке в редакторе сообщений. Если вы

поместите URL-адрес видео на YouTube в отдельной строке, то WordPress обнаружит, что указан URL-адрес Youtube.com, а затем обратится по соответствующему URL-адресу oEmbed, чтобы вставить проигрыватель YouTube в публикацию в веб-интерфейсе вашего сайта. Код для вставки из этого запроса кэшируется в пользовательской записи типа `oembed_cache` и устанавливается как дочерний элемент поста, который вы редактировали.

Пользовательские запросы

Начиная с WordPress 4.9, администраторы могут управлять экспортом личных данных или их удалением от имени пользователей¹. Вкладку, где можно управлять экспортом и удалением пользовательских данных можно найти в меню **Tools** на панели администрирования. Эти запросы хранятся в пользовательской записи типа `user_request`. Подробнее узнать о том, как эти запросы обрабатываются, можно в справочнике WordPress для функции `wp_create_user_request()` ([oreil.ly/R0CC](https://developer.wordpress.org/reference/functions/wp_create_user_request/)).

Повторно используемые блоки

Одна из особенностей нового редактора на основе блоков, представленного в WordPress 5.0, — возможность сохранять блоки, которые вы настраиваете в сообщении, как "повторно используемый блок". Такие блоки хранятся в пользовательском посте типа `wp_block`. Многоразовые блоки описаны в *главе 11*.

Определение и регистрация CPT

Как и стандартные типы записей WordPress, вы можете формировать свои собственные CPT для управления любыми данными, которые вам нужны, в зависимости от того, что вы создаете. Каждый CPT на самом деле просто пост, используемый по-другому. Вы можете зарегистрировать CPT для меню ужина в ресторане, для автомобилей от автодилера, для отслеживания информации о пациенте и документах в кабинете врача, или для почти всего, что только можно представить. В самом деле, любой тип контента, который вы способны придумать, может быть сохранен как пост с вложенными файлами, пользовательскими метаданными и пользовательскими таксономиями.

В нашем примере SchoolPress мы собираемся создать CPT для управления домашним заданием на веб-странице учителя. Наш учитель хочет создать какую-нибудь публикацию, где он может добавлять задания, а его ученики могут найти их на странице класса. Преподаватель также хотел бы иметь возможность загружать дополнительные документы и публиковать комментарии на случай, если у какого-

¹ Новые меры по обеспечению конфиденциальности GDPR были приняты ЕС в 2018 году. Регламент распространяется не только на всех жителей ЕС, но и на все сайты, которые посещают граждане ЕС. Идея разрешить пользователям экспортировать или удалять некритические личные данные хороша вне зависимости от того, где вы живете.

нибудь ученика возникнут вопросы. Как видим, возможности СРТ и здесь пригодятся, не так ли?

Мы можем хранить эту информацию так же, как обрабатываются посты, и отображать их для конечного пользователя в теме, используя тот же цикл `wp_query`, что и с постами.

Функция *register_post_type(\$post_type, \$args)*

Вы можете зарегистрировать СРТ с помощью функции `register_post_type()`. И в большинстве случаев вы регистрируете свой СРТ в файле `functions.php` вашей темы или в файле пользовательского плагина. Эта функция принимает два параметра — имя создаваемого вами поста и массив аргументов:

- ◆ `$post_type` — название вашего СРТ; в нашем примере имя СРТ — "homework" (домашняя работа). Эта строка не должна содержать более 20 символов и не может включать заглавные буквы, пробелы или специальные символы, кроме дефиса или нижнего подчеркивания. Если вы создаете плагин для публичного распространения, то можете использовать префикс в имени вашего СРТ, чтобы избежать конфликтов, на случай если какой-нибудь другой плагин имеет такое же имя;
- ◆ `$args` — это массив множества различных аргументов, которые определяют настройку вашей СРТ.

Вот список всех доступных аргументов и их назначение:

- ◆ `label` — отображаемое имя вашего типа сообщения. В нашем примере — "Homework".
- ◆ `labels` — необязательный массив меток, который можно использовать для описания типа сообщения в пользовательском интерфейсе:
 - `name` — отображаемое имя вашего типа сообщения во множественном числе. Это переопределит аргумент `label`;
 - `singular_name` — название для любого конкретного поста в единственном числе. По умолчанию задано `name`, если другое не указано;
 - `add_new` — по умолчанию задана строка Add New (Добавить новый);
 - `add_new_item` — по умолчанию Add New Post (Добавить новое сообщение);
 - `edit_item` — по умолчанию Edit Post (Редактировать пост);
 - `new_item` — по умолчанию New Post (Новый пост);
 - `view_item` — по умолчанию View Post (Просмотреть пост);
 - `search_items` — по умолчанию Search Posts (Поиск по постам);
 - `not_found` — по умолчанию No Posts Found (Посты не найдены);
 - `not_found_in_trash` — по умолчанию No Posts Found in Trash (Посты в удаленных не найдены);

- `parent_item_colon` — по умолчанию Parent Page (Родительская страница) и используется только для иерархических типов записей;
 - `all_items` — по умолчанию All Posts (Все посты).
- ◆ `menu_name` — название меню для типа сообщения, обычно такое же, как `label` или `labels->name`.
- ◆ `description` — необязательная строка, описывающая тип вашей записи.
- ◆ `publicly_queryable` — необязательный аргумент логического типа (Boolean), который указывает, могут ли запросы по вашему типу записи выполняться в веб-интерфейсе или теме вашего приложения. По умолчанию включен.
- ◆ `exclude_from_search` — необязательный аргумент логического типа (Boolean), который указывает, могут ли ваши типы записей запрашиваться и отображаться в результатах поиска WordPress по умолчанию. Эта функция отключена по умолчанию, поэтому ваши записи будут доступны для поиска.
- ◆ `capability_type` — необязательная строка или массив. Если не указано иное, то по умолчанию будет `post`. Вы можете передать строку существующего типа записи, и новый тип записи, который вы регистрируете, унаследует возможности этого типа записи. Вы также можете определить свой собственный тип возможностей, который будет устанавливать возможности CPT по умолчанию для чтения, публикации, редактирования и удаления. Вы также можете передать массив, если хотите использовать разные слова в единственном и множественном числе. Например, вы можете просто передать строку `"homework"`, так как формы единственного и множественного числа для `"homework"` одинаковы, но когда формы разные (например `"submission"`, `"submissions"`), следует передать массив.
- ◆ `capabilities` — необязательный массив возможностей типа записи, которую вы регистрируете. Этот аргумент вы можете указать вместо `option_type`, если хотите получить более детальный контроль над возможностями, которые вы назначаете для своего нового CPT.

Существует два типа возможностей: *мета* и *примитив*. Метавозможности привязаны к конкретным постам, тогда как примитивные возможности имеют более общее назначение. На практике это означает, что при проверке, имеет ли пользователь метавозможности, вы должны передать параметр `$post_id`:

```
// мета возможности связаны с конкретными постами
if(current_user_can("edit_post", $post_id))
{
    // текущий пользователь может редактировать сообщение с ID = $post_id
}
```

В отличие от метавозможностей, примитивные возможности не проверяются относительно конкретного поста:

```
// примитивные возможности не связаны с конкретными постами
if(current_user_can("edit_posts"))
{
    // текущий пользователь может редактировать публикации в целом
}
```

Возможности, которые могут быть назначены для вашего пользовательского типа сообщения, следующие:

- ♦ `edit_post` — метавозможность пользователя редактировать определенный пост.
- ♦ `read_post` — метавозможность пользователя открыть определенный пост.
- ♦ `delete_post` — метавозможность пользователя удалить определенный пост.
- ♦ `edit_posts` — примитивная возможность пользователя создавать и редактировать публикации.
- ♦ `edit_others_posts` — примитивная возможность пользователя создавать и редактировать чужие публикации.
- ♦ `publish_posts` — примитивная возможность пользователя публиковать посты.
- ♦ `read_private_posts` — примитивная возможность пользователя видеть личные посты.
- ♦ `read` — примитивная возможность пользователя видеть посты.
- ♦ `delete_posts` — примитивная возможность пользователя удалять посты.
- ♦ `delete_private_posts` — примитивная возможность пользователя удалять личные посты.
- ♦ `delete_published_posts` — примитивная возможность пользователя удалять посты.
- ♦ `delete_others_posts` — примитивная возможность пользователя удалять чужие посты.
- ♦ `edit_private_posts` — примитивная возможность пользователя редактировать личные посты.
- ♦ `edit_published_posts` — примитивная возможность пользователя публиковать посты.
- ♦ `map_meta_cap` — задействовать ли внутреннюю обработку метаданных по умолчанию (возможности и роли описаны в *главе 6*). По умолчанию `false`. Вы всегда можете определить свои собственные возможности через `capabilities`; но если вы этого не сделаете, то установка `map_meta_cap` в значение `true` приведет к активизации следующих примитивных возможностей: `capability_type: read, delete_posts, delete_private_posts, delete_published_posts, delete_others_posts, edit_private_posts` и `edit_published_posts`.
- ♦ `hierarchical` — необязательный логический тип, который указывает, может ли публикация быть иерархической и иметь родительский пост. Страницы WordPress настроены так, что вы можете вкладывать одни страницы в другие. Аргумент `hierarchical` по умолчанию отключен.
- ♦ `public` — необязательный аргумент логического типа (Boolean), который указывает, должен ли тип публикации применяться публично или нет в бэкенде или веб-интерфейсе WordPress. По умолчанию — `false`; поэтому без включения этого аргумента и установки его в значение `true` вы не сможете использовать этот

`post_type` в своей теме. Если для `public` установлено значение `true`, то `exclude_from_search`, `publicly_queryable` и `show_ui_nav_menus` автоматически устанавливаются в `true`, если не указано иное.

Большинство CPT будут общедоступными, поэтому они отображаются на веб-интерфейсе или доступны через панель управления WordPress. Другие CPT (например, CPT `Revisions` по умолчанию) обновляются "за кулисами", основываясь на других взаимодействиях с вашим приложением, и в них для `public` будет установлено значение `false`.

- ◆ `rewrite` — необязательный логический тип или массив, предназначенный для создания пользовательской структуры постоянной ссылки (`permalink`) для типа записей. По умолчанию это значение равно `true`, а структура постоянной ссылки для пользовательского сообщения — `/post_type/post_title/`. Если установлено значение `false`, пользовательская постоянная ссылка не будет создана. Вы можете полностью настроить структуру постоянных ссылок сообщения, передав массив со следующими аргументами:
 - `slug` — по умолчанию используется `post_type`, но может быть любая строка, которую вы зададите. Помните, что один и тот же `slug` не может быть в нескольких типах записей, потому что они должны быть уникальными;
 - `with_front` — нужно ли добавлять "префикс" к передней части постоянной ссылки CPT. Если задано значение `true`, слаг для любой страницы, установленный на странице **Settings** → **Reading** панели мониторинга, будет добавлен в постоянную ссылку для записей этого типа;
 - `feeds` — логическое значение, указывающее, может ли тип записи иметь канал RSS. По умолчанию принимает значение аргумента `has_archive`. Если для `feeds` задано значение `false`, каналы не будут доступны;
 - `pages` — логическое значение, которое включает нумерацию страниц для типа записей. Если `true`, страницы архива для этого типа поста будут поддерживать разбиение на страницы;
 - `ep_mask` — EP или `endpoints` (конечные точки) могут быть очень полезны. С помощью этого аргумента вы назначаете маску конечной точки для типа сообщения. Например, мы могли бы установить конечную точку для домашней работы, которая называется "pop-quiz" (викторина). Постоянная ссылка будет выглядеть как `/homework/post-title/pop-quiz/`. В терминологии MVC CPT похож на модуль, и конечные точки могут рассматриваться как различные представления для этого модуля. (Мы рассмотрим конечные точки и другие функции перезаписи в главе 7.)
- ◆ `has_archive` — необязательный логический аргумент или строка, указывающая, может ли тип записи иметь страницу архива. По умолчанию — `false`, вы можете установить значение `true`, если хотите использовать его в своей теме. Файл `archive-<post_type>.php` в вашей теме будет служить для рендеринга страницы архива. Если этот файл недоступен, вместо него будет задействован файл `archive.php` или `index.php`.

- ◆ `query_var` — необязательный логический аргумент или строка, задающая ключ `query_var` для типа записи. Это имя вашего типа записи в базе данных и применяется при написании запросов для работы с этим типом записи. По умолчанию принимает значение аргумента `post_type`. В большинстве случаев вам не нужно, чтобы ваш `query_var` и ваш `post_type` были разными, но вы можете представить длинное имя типа записи, например `"directory_entry"`, для которого вы хотели бы задать более короткий фрагмент, такой как `"dir"`.
- ◆ `supports` — необязательный логический аргумент или массив, который указывает, какие функции метаполей будут доступны для новой публикации или странице редактирования сообщения. По умолчанию передается массив с аргументами заголовков и редактор. Далее приведен список всех доступных аргументов (чтобы использовать одну из этих функций с вашим СРТ, убедитесь, что она включена в массиве `supports`):
 - `title;`
 - `editor;`
 - `comments;`
 - `revisions;`
 - `trackbacks;`
 - `author;`
 - `excerpt;`
 - `page-attributes;`
 - `thumbnail;`
 - `custom-fields;`
 - `post-formats.`
- ◆ `register_meta_box_cb` — необязательная строка, которая позволяет вам предоставить пользовательскую функцию обратного вызова для интеграции ваших собственных пользовательских метаполей.
- ◆ `permalink_epmask` — необязательная строка для указания того, какие типы конечных точек вы хотите связать с пользовательским типом записи. Маской перезаписи конечной точки по умолчанию является `EP_PERMA LINK`. Дополнительная информация о конечных точках приведена в *главе 7*.
- ◆ `taxonomies` — необязательный массив, который задает произвольные встроенные (категории и теги) или пользовательские таксономии, которые вы хотите связать с типом записи. По умолчанию ссылки на таксономии отсутствуют. Подробнее о таксономиях см. разд. "Создание пользовательских таксономий".
- ◆ `show_ui` — необязательный аргумент типа Boolean, который указывает, будет ли доступен базовый пользовательский интерфейс публикации для нового типа записи в бэкенде. По умолчанию принимает значение аргумента `public`. Если `show_ui` имеет значение `false`, то вы не сможете заполнять свои посты из области администрирования бэкенда.



Рекомендуется установить для `show_ui` значение `true`, даже для CPT, которые обычно не добавляются и не редактируются через панель администратора. Например, плагин bbPress добавляет темы и ответы в виде CPT, которые добавляются и редактируются через пользовательский интерфейс форума. Однако `show_ui` имеет значение `true`, предоставляя другой интерфейс, из которого администраторы могут искать темы и ответы, просматривать их и управлять ими.

- ◆ `menu_position` — необязательный целочисленный аргумент, который определяет положение типа записи в меню, которое слева.

В кодексе WordPress представлен хороший список наиболее часто используемых положений (bawawwp.com/posttype-ui), он поможет вам определить, где разместить пункт меню для вашей пользовательской записи:

- 5: под публикациями;
- 10: под медиа;
- 15: под ссылками;
- 20: под страницами;
- 25: под комментариями;
- 60: под первым разделителем;
- 65: под плагинами;
- 70: под пользователями;
- 75: под инструментами;
- 80: под настройками;
- 100: под вторым разделителем.

- ◆ `menu_icon` — необязательная строка URL-адреса для пользовательского значка, которая может представлять тип записи.
- ◆ `can_export` — необязательный аргумент типа `Boolean`, указывающий, можно ли экспортировать тип записи через экспортер WordPress в меню **Tools** → **Export**. Этот аргумент по умолчанию имеет значение `true`, что позволяет администратору экспортировать.
- ◆ `show_in_nav_menus` — необязательный аргумент типа `Boolean`, который указывает, можно ли добавлять сообщения данного типа записи в пользовательское меню навигации в разделе **Appearance** → **Menus**. По умолчанию принимает значение аргумента `public`.
- ◆ `show_in_menu` — необязательный аргумент типа `Boolean` или строка, указывающая, отображать ли тип записи в административном меню и, возможно, где его показывать. Если установлено значение `true`, тип записи отображается в меню как отдельный элемент. Если установлено `false`, пункт меню для данного типа записи не отображается. Вы также можете передать строку имени любого другого пункта меню. При этом тип записи помещается в подменю переданного пункта меню. По умолчанию принимает значение аргумента `show_ui`.

- ◆ `show_in_admin_bar` — необязательный аргумент типа Boolean, который указывает, доступен ли тип записи в панели администратора WordPress. По умолчанию принимает значение аргумента `show_in_menu`.
- ◆ `delete_with_user` — необязательный аргумент типа Boolean, который указывает, следует ли удалять все сообщения для типа записи, созданного данным пользователем. Если установлено значение `true`, то сообщения, созданные пользователем, будут перемещены в корзину при удалении пользователя. Если установлено значение `false`, то сообщения не будут перемещаться в корзину при удалении пользователя. По умолчанию сообщения перемещаются в корзину, если в аргументе `post_type_supports` указан автор. Если нет, то сообщения не перемещаются в корзину.
- ◆ `show_in_rest` — необязательный аргумент типа Boolean, который указывает, доступен ли этот CPT через REST API. Значение по умолчанию — `false`. API REST описан в *главе 10*.
- ◆ `rest_base` — необязательная строка для изменения базового слага, используемая при доступе к CPT этого типа через REST API. Значением по умолчанию для этого аргумента является имя типа записи, заданное в качестве первого параметра функции `register_post_type()`.
- ◆ `rest_controller_class` — необязательная строка для изменения контроллера при доступе к этому типу записи через REST API. Этот аргумент должен иметь значение `WP_REST_Posts_Controller` или имя класса PHP, который наследует класс `WP_REST_Posts_Controller`. Значение по умолчанию — `WP_REST_Posts_Controller`.
- ◆ `_builtin` — этот аргумент вам не потребуется изменять. Он по умолчанию задан для типов сообщений WordPress, чтобы отличать их от CPT.
- ◆ `_edit_link` — URL-адрес ссылки для редактирования записи. Это также для внутреннего использования, и вам не нужно его менять. Если вы хотите изменить страницу, на которую указывает ссылка, чтобы отредактировать запись, используйте фильтр `get_edit_post_link`, который передает ссылку для редактирования по умолчанию вместе с идентификатором записи.

Листинг 5.1 иллюстрирует регистрацию новых CPT "Homework" и "Submissions" с помощью функции `register_post_type()`, код которой можно найти в файле `wp-includes/post.php`. Обратите внимание, что в нашем примере мы используем только несколько из множества доступных аргументов.

Листинг 5.1. Регистрация CPT

```
<?php
// пользовательская функция для регистрации типа записи "homework"
function schoolpress_register_post_type_homework() {
    register_post_type('homework',
        array(
            'labels' => array(
```

```

        'name' => __('Homework'),
        'singular_name' => __('Homework')
    ),
    'public' => true,
    'has_archive' => true,
)
);
}
// вызываем нашу пользовательскую функцию с помощью хука init
add_action('init', 'schoolpress_register_post_type_homework');

// пользовательская функция для регистрации типа записи "submissions"
function schoolpress_register_post_type_submission() {
    register_post_type('submissions',
        array(
            'labels' => array(
                'name' => __('Submissions'),
                'singular_name' => __('Submission')
            ),
            'public' => true,
            'has_archive' => true,
        )
    );
}
// вызываем нашу пользовательскую функцию с помощью хука init
add_action('init', 'schoolpress_register_post_type_submission');
?>

```

Если вы вставите код листинга 5.1 в файл `functions.php` вашей активной темы или активного плагина, то в панели администрирования WordPress под пунктом меню "Comments" появятся два новых пункта меню, которые называются "Homework" и "Submissions".



Если вы устали от написания функций для регистрации различных пользовательских типов записей, то можете использовать замечательный плагин под названием Custom Post Type U (bwawwp.com/posttype-ui).

Что такое таксономия и как ее использовать?

Мы уже упоминали таксономии в *главе 2*, но что конкретно представляет собой таксономия? Таксономии группируют сообщения по терминам. Возьмем, к примеру, категории сообщений и теги сообщений — это просто встроенные таксономии, прикрепленные к стандартному типу записей "пост". Вы можете определить столько пользовательских таксономий или категорий, сколько хотите, и распределить их

по нескольким типам записей. Например, мы можем создать специальную таксономию "Subject", которая включает в себя все школьные предметы в качестве терминов и привязана к нашему CPT "Homework".

Таксономии и метаданные постов

Один из вопросов, с которым вы часто будете сталкиваться, когда хотите прикрепить биты данных к сообщениям, — это, использовать ли таксономию или поле метазаписи (или оба варианта). Как правило, термины, объединяющие разные посты, должны кодироваться как таксономии, а данные, относящиеся к каждому отдельному посту, должны кодироваться как метаполя постов.

Метаполя постов хорошо работают для данных, относящихся к отдельным сообщениям, и не предназначены для группировки сообщений вместе. В SchoolPress имеет смысл кодировать такие вещи, как "требуемая длина ответа на задание" (например, 500 слов), в качестве метаполя. На практике будет встречаться несколько ответов разной длины, но нам никогда не понадобится "получать все задания, требующие 500 слов". Таким образом, для этой информации достаточно метаполя.

Таксономии хороши для данных, которые служат для группировки сообщений вместе. В SchoolPress в качестве таксономии имеет смысл кодировать такие вещи, как предмет задания (например, математика или английский язык). В отличие от длины ответа, мы хотим выполнять запросы типа "получить все задания по математике". Это легко сделать с помощью запроса таксономии. Более важно, что такие запросы выполняются быстрее для данных таксономии, чем для метаполей.

Почему запросы таксономии обычно быстрее? Метаполя хранятся в таблице `wp_postmeta`. Если бы мы сохраняли дату выполнения задания как метаполе поста, это выглядело бы так:

meta_id	post_id	meta_key	meta_value
1	1	due_date	2018-09-07
2	2	due_date	2018-09-14

Столбцы `meta_id`, `post_id` и `meta_key` индексируются, а столбец `meta_value` — нет. Это означает, что запросы типа "получить срок выполнения для этого задания" будут выполняться быстро, но запросы типа "получить все задания, которые должны быть выполнены в 2018-09-07" будут выполняться медленнее, особенно, если у вас большой сайт с множеством данных, накопленных в таблице `wp_postmeta`. Ключ `meta_value` является единственным столбцом в `wp_postmeta` без индекса, поскольку добавление индекса здесь значительно увеличит как объем памяти, необходимый для этой таблицы, так и время вставки. На практике у сайта будет много разных метазначений, а набор идентификаторов записей и метаключей для создания индексов окажется гораздо меньше.

Если вы сохранили даты выполнения заданий в пользовательской таксономии, то запрос "получить все задания, назначенные на эту дату" будет выполняться намного быстрее. Каждый конкретный срок должен быть термином в таблице `wp_terms` с соответствующей записью в таблице `wp_terms_taxonomy`. Таблица

`wp_terms_relationships`, которая прикрепляет термины к сообщениям, содержит индексруемые поля `object_id` (сообщения здесь являются объектами) и `term_taxonomy_id`. Так что "получить все сообщения с этим `term_taxonomy_id`" — это быстрый запрос.

Если вы просто хотите отобразить дату выполнения на странице задания, то должны сохранить ее в метаполях публикации. Если вы хотите предоставить отчет обо всех заданиях, причитающихся к определенной дате, вам следует рассмотреть возможность добавления таксономии для отслеживания сроков исполнения.

С другой стороны, из-за характера сроков исполнения (у вас может быть 365 новых терминов в год), использование таксономии для них может оказаться излишним. В вашей базе данных окажется много бесполезных терминов, отслеживающих, какие задания были выполнены два года назад.

Кроме того, в этом конкретном случае увеличение скорости может быть незначительным, поскольку отчет о дате выполнения предназначен для поднабора заданий в пределах определенной группы классов. На практике мы не будем запрашивать задания по дате выполнения по всей таблице `wp_postmeta`. Мы отфильтруем запрос, чтобы он выполнялся только для сообщений о заданиях для определенного класса. В таблице `wp_postmeta` для сайта SchoolPress в целом (сотни школ, тысячи учителей и классов) могут оказаться миллионы заданий, но будет только несколько заданий для конкретного класса или группы классов, в которых состоит студент.

Другое соображение при выборе между метаполями и таксономиями заключается в учете того, как пользователи будут управлять этими данными. Если поле будет присутствовать только на бэкенде и у вас нет проблем со скоростью запросов, то сохранение значения в метаданных поста так же просто, как один вызов функции `update_post_meta()`. Если вы хотите, чтобы администраторы могли создавать новые термины, напишите их описание, создайте иерархию и используйте раскрывающиеся списки или флажки для связи их с постами. Итак мы только что описали, что вы получаете бесплатно при регистрации таксономии. При наличии метаполей поста вам необходимо будет построить собственный интерфейс.

Наконец, я упоминал ранее, что бывают случаи, когда вы хотите задействовать как метаполе, так и таксономию для отслеживания одного фрагмента данных. Примером этого в контексте приложения SchoolPress может быть поиск учебника и главы для назначения задания. Представьте, что вам нужен отчет для студента со всеми его заданиями, организованными по учебникам и упорядоченными по главам в этих книгах. Поскольку вы хотите, чтобы учителя могли управлять учебниками как терминами в меню администратора, а вам нужно выполнять запросы типа "получить все задания для этого учебника", имеет смысл хранить учебники в пользовательской таксономии.

С другой стороны, главы из учебника могут быть сохранены в метаданных поста. Главы часто встречаются в книгах и заданиях, но нет смысла запрашивать "все задания для главы 1" в разных учебниках. Поскольку мы сможем выполнить предварительную фильтрацию, чтобы получить все задания по учебнику или по студенту,

то через метаполе главы или, возможно, метаполе `textbook_chapter` с данными типа "PrinciplesOfMath.Ch1", сможем упорядочить назначения в отчете.

Фуф... теперь, когда мы уяснили, где требуются таксономии, давайте выясним, как их создавать.

Создание пользовательских таксономий

Вы можете зарегистрировать свои собственные таксономии с помощью функции `register_taxonomy()`, которая находится в файле `wp-includes/taxonomy.php`.

Функция

`register_taxonomy($taxonomy, $object_type, $args)`

Функция `register_taxonomy()` принимает три параметра:

- ◆ `$taxonomy` — обязательная строка имени объекта таксономии. В нашем примере название нашей таксономии — "subject".
- ◆ `$object_type` — обязательный массив или строка СРТ, к которым вы присоединяете эту таксономию. В нашем примере мы используем строку и присоединяем таксономию предмета к типу записи "homework". Мы можем установить более одного типа записи, передав массив имен типов записей.
- ◆ `$args` — необязательный массив из множества аргументов определяет, как построена ваша пользовательская таксономия. Обратите внимание, что в нашем примере мы указали только некоторые из множества доступных аргументов, которые можно передать в функцию `register_taxonomy()`.

Вот полный список доступных аргументов:

- ◆ `label` — необязательная строка отображаемого имени вашей таксономии.
- ◆ `labels` — необязательный массив меток, предназначенный для описания таксономии в пользовательском интерфейсе:
 - `name` — отображаемое имя вашей таксономии во множественном числе. Оно переопределит аргумент `label`;
 - `singular_name` — название для одного объекта этой таксономии. По умолчанию `Category` (Категория);
 - `search_items` — по умолчанию `Search Categories` (Поиск по категориям);
 - `popular_items` — эта строка не используется в иерархических таксономиях. По умолчанию `Popular Tags` (Популярные теги);
 - `all_items` — по умолчанию `All Categories` (Все категории);
 - `parent_item` — эта строка используется только в иерархических таксономиях. По умолчанию `Parent Category` (Родительская категория);
 - `parent_item_colon` — то же, что и аргумент `parent_item`, но с двоеточием в конце;

- `edit_item` — по умолчанию `Edit Category` (Редактировать категорию);
 - `view_item` — по умолчанию `View Category` (Просмотреть категорию);
 - `update_item` — по умолчанию `Update Category` (Обновить категорию);
 - `add_new_item` — по умолчанию `Add New Category` (Добавить новую категорию);
 - `new_item_name` — по умолчанию `New Category Name` (Новое имя категории);
 - `separate_items_with_commas` — эта строка используется только в неиерархических таксономиях. По умолчанию `Separate tags with commas` (Разделять теги запятыми);
 - `add_or_remove_items` — эта строка встречается только в неиерархических таксономиях. По умолчанию `Add or remove tags` (Добавить или удалить теги);
 - `choose_from_most_used` — эта строка используется только в неиерархических таксономиях. По умолчанию `Choose from the most used tags` (Выбрать из наиболее часто используемых тегов).
- ◆ `hierarchical` — необязательный аргумент логического типа, который указывает, является ли таксономия иерархической или может ли термин таксономии иметь родительские или дочерние термины. Это похоже на таксономию категорий по умолчанию, а неиерархические таксономии похожи на таксономию тегов по умолчанию. Значение по умолчанию — `false`.
 - ◆ `update_count_callback` — необязательная строка, которая работает как хук. Он вызывается при обновлении счетчика соответствующего типа записи.
 - ◆ `rewrite` — необязательный логический аргумент или массив, предназначенный для настройки структуры постоянной ссылки таксономии. Значение перезаписи по умолчанию — слаг таксономии.
 - ◆ `query_var` — необязательный аргумент типа `Boolean` или строка, которую можно использовать для настройки выражения `query_var ? $query_var=$term`. Значение по умолчанию — имя таксономии.
 - ◆ `public` — необязательный аргумент типа `Boolean`, который указывает, следует ли публично использовать таксономию в бэкенде или веб-интерфейсе WordPress. По умолчанию этот аргумент имеет значение `false`, поэтому без включения этого аргумента и установки его в значение `true` вы не сможете использовать эту таксономию в своей теме. Если для `public` установлено значение `true`, для `show_ui`, `publicly_queryable` и `show_in_nav_menus` автоматически устанавливается значение `true`, если не задано иное.
 - ◆ `publicly_queryable` — необязательный аргумент типа `Boolean`, который указывает, может ли таксономия быть публично запрошена из внешнего интерфейса. По умолчанию установлено значение `true`.
 - ◆ `show_ui` — необязательный аргумент типа `Boolean`, который определяет, будет ли таксономия иметь интерфейс администратора, аналогичный интерфейсу категорий или тегов. По умолчанию принимает значение аргумента `public`.

- ◆ `show_in_nav_menus` — необязательный логический параметр, указывающий, будет ли таксономия доступна в навигационных меню. По умолчанию принимает значение аргумента `public`.
- ◆ `show_in_rest` — необязательный аргумент типа `Boolean`, который определяет, доступна ли таксономия через REST API. Значение по умолчанию — `false`. API REST описан в *главе 10*.
- ◆ `rest_base` — необязательная строка для изменения базового слага, используемая при доступе к СРТ этой таксономии через REST API. Значение по умолчанию — имя таксономии, заданное в качестве первого параметра функции `register_taxonomy()`.
- ◆ `rest_controller_class` — необязательная строка для изменения контроллера, используемого при доступе к постам данной таксономии через REST API. Этот аргумент должен иметь значение `WP_REST_Terms_Controller` или имя класса PHP, который наследует класс `WP_REST_Terms_Controller`. Значение по умолчанию — `WP_REST_Terms_Controller`.
- ◆ `show_tagcloud` — необязательный аргумент типа `Boolean`, который указывает, может ли таксономия быть включена в виджет Tag Cloud. По умолчанию принимает значение аргумента `show_ui`.
- ◆ `show_admin_column` — необязательный аргумент типа `Boolean`, который указывает, будет ли создан новый столбец для вашей таксономии для типа записи, к которому он прикреплен, на странице редактирования/списка типов записей на бэкенде. По умолчанию `false`.
- ◆ `capabilities` — необязательный массив возможностей для этой таксономии со значением по умолчанию `none`. Вы можете передать следующие аргументы и/или любые пользовательские возможности:
 - `manage_terms`;
 - `edit_terms`;
 - `delete_terms`;
 - `assign_terms`.

В следующем нашем примере типа записи `homework` мы собираемся создать таксономию под названием "Subject", чтобы у нас был термин для каждого предмета, такого как математика, естествознание, иностранный язык и социальные науки.

Пример

```
<?php
// пользовательская функция для регистрации таксономии "Предмет"
function schoolpress_register_taxonomy_subject() {
    register_taxonomy(
        'subject',
        'homework',
```

```

        array(
            'label' => __('Subjects'),
            'rewrite' => array('slug' => 'subject'),
            'hierarchical' => true
        )
    );
}

// вызываем нашу пользовательскую функцию с помощью хука init
add_action('init', 'schoolpress_register_taxonomy_subject');
?>

```

Обратите внимание, что в коде этого примера таксономия предмета настроена как категория в сообщении, поскольку ее иерархический аргумент имеет значение `true`. Вы можете создать столько предметов, сколько захотите, и вкладывать их друг в друга.

В разделе **Homework** → **Subjects** на бэкенде вы можете добавить свои термины так же, как вы бы добавили новые категории в публикации.

Функция `register_taxonomy_for_object_type($taxonomy, $object_type)`

Как поступить, если вы хотите использовать таксономию по умолчанию для СРТ? Если вам потребуется та же таксономия тегов, которая прикреплена к типу записи публикаций, в нашем типе записи `homework`, то здесь поможет функция `register_taxonomy_for_object_type()`, позволяющая прикрепить любые таксономии к любым типам записей. Эта функция также находится в файле `wp-includes/taxonomy.php`.

Функция `register_taxonomy_for_object_type()` принимает два параметра:

- ◆ `$taxonomy` — обязательная строка имени объекта таксономии.
- ◆ `$object_type` — обязательная строка с названием типа записи, к которому вы хотите прикрепить свою таксономию.

В следующем примере мы присоединяем таксономию тегов по умолчанию к нашему типу записи `homework`.

Пример

```

<?php
function schoolpress_register_taxonomy_for_object_type_homework(){
    register_taxonomy_for_object_type('post_tag', 'homework');
}

add_action('init', 'schoolpress_register_taxonomy_for_object_type_homework');
?>

```

Если вы запустите этот пример, то обратите внимание, что таксономия "тегов" теперь доступна в пункте меню **Homework**. Плагин Custom Post Types UI (bwwwp.com/posttype-ui) также имеет пользовательский интерфейс для создания и управления пользовательскими таксономиями.

Использование CPT и таксономий в ваших темах и плагинах

В следующих разделах мы рассмотрим некоторые моменты, которые следует учитывать при использовании CPT в ваших темах или плагинах.

Тема архива и файлы шаблона Single

Большинство тем WordPress имеют файл `archive.php`, который отображает ваши сообщения на странице `archive/listing`, и файл `single.php`, который отвечает за отображение информации в одном сообщении. Вы можете легко создать общий архив и отдельные файлы для ваших зарегистрированных CPT.

Сделайте копию `archive.php` и назовите ее `archive-homework.php`. Теперь вы должны автоматически получить страницу со списком всех ваших домашних заданий в том же формате, что и обычная страница архива сообщений (по адресу `domain.com/homework/`).

Вы можете применить тот же метод к файлу `single.php`. Скопируйте его и назовите `single-homework.php`. В результате у вас должна быть отдельная страница для каждого домашнего задания (по адресу `domain.com/homework/science-worksheet/`). Теперь вы можете изменить разметку архива CPT или файла `single.php`, чтобы отображать ваши данные не так как ваши публикации в блоге.



Чтобы использовать собственный файл архива, вы должны установить аргумент `has_archive` при регистрации своего пользовательского типа записи в значение `true`. Аргумент `has_archive` является частью функции `register_post_type()`.

Старый добрый класс `WP_Query` и метод `get_posts()`

В некоторых случаях создание архива и файла `single.php` для вашего собственного типа записи может быть недостаточно для обеспечения требуемой пользовательской функциональности. Как быть, если вы хотите просмотреть все сообщения определенного типа в виджете боковой панели или в шорткоде на странице? С помощью класса `WP_Query` или метода `get_posts()` вы можете установить для параметра `post_type` значение запроса и циклически проходить по вашим записям CPT так же, как с обычными публикациями.

В листинге 5.2 мы создадим форму отправки домашнего задания под любым контентом отдельной публикации с типом записи `homework`.

```

<?php
function schoolpress_the_content_homework_submission($content){
global $post, $current_user;
// Не делать этого для любого другого типа поста, кроме домашней работы.
if (! is_single() || $post->post_type != 'homework')
return $content;
// Не делать этого, если пользователь не вошел в систему.
if (! is_user_logged_in())
return $content;
// Проверка, что текущий пользователь уже отправлял
// это домашнее задание.
$submissions = get_posts(array(
'post_author' => $current_user->ID,
'posts_per_page' => '1',
'post_type' => 'submissions',
'meta_key' => '_submission_homework_id',
'meta_value' => $post->ID
));
foreach ($submissions as $submission) {
$submission_id = $submission->ID;
}
// Обработка отправки формы, если пользователь еще не сделал это задание.
if (!$submission_id &&
isset($_POST['submit-homework-submission']) &&
isset($_POST['homework-submission'])) {
$submission = $_POST['homework-submission'];
$post_title = $post->post_title;
$post_title .= ' - Submission by ' . $current_user->display_name;
// Вставляем задания текущих пользователей как пост в наш
// CPT заданий.
$args = array(
'post_title' => $post_title,
'post_content' => $submission,
'post_type' => 'submissions',
'post_status' => 'publish',
'post_author' => $current_user->ID
);
$submission_id = wp_insert_post($args);
// Добавить метазапись, чтобы связать отправленное задание с домашней работой.
add_post_meta($submission_id, '_submission_homework_id',
$post->ID);
// Создать собственное сообщение.
$message = __(
'Ваша домашняя работа была отправлена и ожидает рассмотрения.',

```

```

'schoolpress'
);
$message = '<div class="homework-submission-message">' . $message .
'</div>';
// Вывести сообщение перед отфильтрованной переменной $content.
$content = $message . $content;
}
// Добавить ссылку на файлы пользователя, если задание уже было выполнено.
if($submission_id) {
$message = sprintf(__(
'Нажмите %s здесь %s , чтобы посмотреть ваше отправленное задание
assignment.',
'schoolpress'),
'<a href="' . get_permalink($submission_id) . '">',
'</a>');
$message = '<div class="homework-submission-link">' . $message .
'</div>';
$content .= $message;
// Добавить базовую форму отправки после фильтрации переменной $content.
} else {
ob_start();
?>
<h3><?php _e('Прикрепите выполненное задание ниже!', 'schoolpress');?></h3>
<form method="post">
<?php
wp_editor('', 'homework-submission', array('media_buttons' => false));
?>
<input type="submit" name="submit-homework-submission" value="Submit" />
</form>
<?php
$form = ob_get_contents();
ob_end_clean();
$content .= $form;
}
return $content;
}
// Добавить фильтр в the_content, чтобы мы могли выполнить наш собственный код для
работы с домашними заданиями
add_filter('the_content', 'schoolpress_the_content_homework_submission', 999);
?>

```

Вы, наверное, заметили, что мы еще не рассматривали некоторые функции:

- ♦ `ob_start()` — эта функция PHP включает буферизацию вывода. Пока активна буферизация вывода, вывод не отправляется в браузер; вместо этого выходные данные сохраняются во внутреннем буфере.

- ◆ `wp_editor()` — эта функция WordPress выводит тот же редактор WYSIWYG, который вы видите при добавлении или редактировании поста. Вы можете вызывать эту функцию везде, где хотите вставить редактор. Мы подумали, что форма отправки домашней работы будет идеальным местом. Мы рассмотрим все параметры этой функции в *главе 7*.
- ◆ `ob_get_contents()` — мы установили переменную с именем `$form` для этой функции PHP. В результате весь контент между вызовом функции `ob_start()` и этой функции превращается в переменную с именем `$form`.
- ◆ `ob_end_clean()` — эта функция PHP очищает выходной буфер и отключает выходную буферизацию.

Мы использовали эти функции в такой последовательности, потому что функция `wp_editor()` в настоящее время не имеет аргумента для возврата редактора как переменной и вывода его в браузер при вызове. При отсутствии указанных функций мы не смогли бы отобразить наш редактор после того, как переменная `$content` была передана в фильтр `the_content`.

В следующем примере мы позаботимся о том, чтобы только администраторы имели доступ ко всем отправленным домашним заданиям, а другие пользователи могли получить доступ только к заданиям, которые они выполнили сами.

Пример

```
<?php
function schoolpress_submissions_template_redirect(){
global $post, $user_ID;
// запускаем эту функцию только для типа записей отправки заданий
if ($post->post_type != 'submissions')
return;
// проверка, является ли post_author текущим идентификатором пользователя
if ($post->post_author == $user_ID)
$no_redirect = true;
// проверяем, является ли текущий пользователь администратором
if (current_user_can('manage_options'))
$no_redirect = true;
// если $no_redirect=false, то перенаправляем на домашнюю страницу
if (! $no_redirect) {
wp_redirect(home_url());
exit();
}
}
// Используем хук template_redirect для вызова функции, которая решает,
// может ли текущий пользователь получить доступ к отправленному заданию.
add_action('template_redirect', 'schoolpress_submissions_template_redirect');
?>
```

Метаданные и СРТ

Вы можете использовать те же метафункции публикаций, которые мы подробно рассмотрели в *главе 2*, с любым созданным вами СРТ. Получение, добавление, обновление и удаление метаданных постов работает для всех типов записей.

Если вы зарегистрировали СРТ и добавили настраиваемые поля в аргументе `support`, по умолчанию при добавлении нового сообщения или редактировании сообщения такого типа, вы увидите метаполе **Custom Fields**. Возможно, вы уже знакомы с этим метаполем; это очень простая форма, предназначенная для хранения метаданных, прикрепленных к сообщению.



Если вы не видите раздел **Custom Fields** на странице редактирования сообщения, то вам, вероятно, необходимо включить эту возможность. Если вы используете Block Editor, нажмите кнопку **More Tools** (три точки в правом верхнем углу). В нижней части страницы нажмите кнопку **Options**, а затем в разделе **Advanced Panels** найдите строку **Custom Fields**. Если вы работаете в классическом редакторе, то в правом верхнем углу откройте вкладку **Screen Options** и найдите флажок **Custom Fields**.

Что делать, если вам понадобится более удобный пользовательский интерфейс для добавления метаданных на сервер? Выход из этой ситуации — создание пользовательского метаполя.

Функция `add_meta_box($id, $title, $callback, $screen, $context, $priority, $callback_args)`

Функция `add_meta_box()` добавит метаполе на один или несколько экранов:

- ◆ `$id` — обязательная строка — уникальный идентификатор метаполя, которое вы создаете;
- ◆ `$title` — обязательная строка заголовка или видимого имени метаполя, которое вы создаете;
- ◆ `$callback` — обязательная строка — имя функции, которая вызывается для вывода HTML-кода внутри создаваемого вами метаблока;
- ◆ `$screen` — необязательный параметр — экран или экраны, где появится ваше метаполе. Принимает идентификатор экрана, объект `WP_Screen` или массив идентификаторов экрана. Значение по умолчанию `null`;
- ◆ `$context` — необязательная строка контекста на странице, где должно отображаться ваше метаполе (обычный — `normal`, расширенный — `advanced`, боковой — `side`). По умолчанию — `расширенный`;
- ◆ `$priority` — необязательная строка — приоритет в контексте, где должны отображаться поля (высокий, низкий);
- ◆ `$callback_args` — необязательный массив аргументов, передаваемый в функцию обратного вызова, на которую вы указали с помощью параметра `$callback`. Ваша функция обратного вызова автоматически получит объект `$post` и любые другие аргументы, которые вы здесь зададите.



Здесь мы фокусируемся на том, как добавить метаполя на экран редактирования записи, однако, метаблоки можно использовать на любом экране администратора. Чтобы применить метаполя на своей странице, задайте уникальное значение для параметра экрана затем вызовите функцию `do_meta_boxes()` (oreil.ly/31Mdp) для этого экрана в функции обратного вызова вашей страницы. Это хорошо подходит для страниц отчетов или других страниц, на которых вы хотите, чтобы пользователи могли скрывать поля, переставлять их или добавлять новые с помощью пользовательских хуков.

В листинге 5.3 мы собираемся создать собственный метаблок для всех сообщений нашего типа записей `homework`. Это метаполе будет содержать флажок, указывающий, требуется ли отправка домашнего задания, а также селектор даты для даты выполнения домашнего задания.

Листинг 5.3. Пользовательское метаполе

```
<?php
// Обратный вызов для добавления пользовательского метаблока. функция
function schoolpress_homework_add_meta_boxes(){

    add_meta_box(
        'homework_meta',
        'Additonal Homework Info',
        'schoolpress_homework_meta_box',
        'homework',
        'side'
    );
}

// Используйте хук add_meta_boxes, чтобы вызвать пользовательскую
// функцию для добавления нового метаполя.
add_action('add_meta_boxes', 'schoolpress_homework_add_meta_boxes');

// Это функция обратного вызова, вызываемая из add_meta_box.
function schoolpress_homework_meta_box($post){
    // Используем здесь 2 строки, чтобы URL поместился в книге ;)
    $smoothness_url = 'http://ajax.googleapis.com/ajax/libs/';
    $smoothness_url.= 'jqueryui/1.12.1/themes/smoothness/jquery-ui.css';

    // Активируем jquery date picker.
    wp_enqueue_script('jquery-ui-datepicker');
    wp_enqueue_style('jquery-style', $smoothness_url);

    // Установить метаданные, если они уже существуют.
    $is_required = get_post_meta($post->ID,
        '_schoolpress_homework_is_required', 1);
```

```

$due_date = get_post_meta($post->ID,
    '_schoolpress_homework_due_date', 1);
// Вывод полей метаданных.
?>
<p>
<input type="checkbox"
name="is_required" value="1" <?php checked($is_required, '1'); ?>>
This assignment is required.
</p>
<p>
Due Date:
<input type="text"
name="due_date" id="due_date" value="<?php echo $due_date;?>">
</p>
<script>
// Присоединяем jQuery для выбора даты к нашему полю due_date.
jQuery(document).ready(function() {
    jQuery('#due_date').datepicker({
        dateFormat : 'mm/dd/yy'
    });
});
</script>
<?php
}

// Обратный вызов для сохранения пользовательских метаданных в базе данных.
function schoolpress_homework_save_post($post_id){

    // Не сохраняйте ничего, если в WP настроено автосохранение.
    if (defined('DOING_AUTOSAVE') && DOING_AUTOSAVE)
        return $post_id;

    // Проверка правильности типа сообщения и наличия
    // у пользователя нужных разрешений.
    if ('homework' == $_POST['post_type']) {

        if (! current_user_can('edit_page', $post_id))
            return $post_id;

    } else {

        if (! current_user_can('edit_post', $post_id))
            return $post_id;

    }

    // Обновление метаданных домашней работы
    update_post_meta($post_id,

```

```

        '_schoolpress_homework_is_required',
        $_POST['is_required']
    );
    update_post_meta($post_id,
        '_schoolpress_homework_due_date',
        $_POST['due_date']
    );
}
// Вызываем пользовательскую функцию для обработки сохранения наших метаданных.
add_action('save_post', 'schoolpress_homework_save_post');
?>

```

Если вы квалифицированный разработчик, то, вероятно, подумаете: где же *nonce* (однократно используемое число)? Почему значения `$_POST` не очищены? Если эти вопросы у вас не возникают, то вам следует задуматься, потому что безопасность очень важна! Если вы не знаете, о чем мы говорим, обратитесь к *главе 8*, где более подробно будут рассмотрены эти передовые практики (и однократно используемые числа). Здесь мы намеренно пропустили этот дополнительный код, чтобы пример был коротким и удобным для изучения, но мы знаем, что когда вы пишете собственную программу, вы всегда должны использовать *nonce* и очищать ваши данные.



При создании метаблоков и пользовательских метаполей мы рекомендуем плагин CMB2 (github.com/CMB2/CMB2). Вы можете легко включить CMB2 в свою тему или в любой пользовательский плагин, чтобы быстро и легко создавать собственные метаблоки и метаполя внутри них.

Использование метаблоков в редакторе блоков Block Editor

Функция `add_meta_box()` работает как с редактором блоков, так и с классическим редактором. В обоих случаях метаблоки, добавленные с "боковым" контекстом, будут отображаться на правой боковой панели экрана редактирования сообщения. В классическом редакторе все метаблоки боковой панели всегда видны². В редакторе блоков метаполя боковой панели отображаются на вкладке **Document** на правой боковой панели.

В обеих версиях под основной областью редактора будут отображаться метаблоки, добавленные с обычным или расширенным контекстом.

Поскольку вы также можете использовать блоки для обновления метаданных поста, некоторые метаблоки могут работать лучше как поля. Мы более подробно рассмот-

² Несмотря на то, что нажатие на заголовок метаблока, где бы он ни отображался, скрывает или показывает полное содержимое метаблока.

рим блоки в *главе 11*, но далее приведены некоторые вопросы, которые вы должны задать себе при принятии решения о том, следует ли разрабатывать определенное поле или функцию как пользовательский тип блока или как метаблок. Ответы здесь не являются жесткими правилами, но их следует воспринимать как предложение склоняться к одной реализации, а не к другой.

- ◆ Должны ли эти метаданные быть установлены для каждого сообщения такого типа?

Используйте метаполе с "нормальным" контекстом, чтобы оно отображалось под редактором публикации. Как альтернатива, вы можете совместить блок с шаблоном блока, чтобы каждый новый пост содержал блок по умолчанию. Подробнее о шаблонах блоков сказано в *главе 11*.

- ◆ Подойдет ли размещение элементов управления для этих метаданных на боковой панели?

Если нет, задайте метаполе с контекстом "normal", чтобы отображаться под редактором публикации. Для ваших форм и полей будет больше места. Метаблок "Require Membership" в Paid Memberships Pro с одним лишь набором флажков прекрасно вписывается в боковую панель. Поля ценообразования WooCommerce с собственными вкладками лучше всего располагаются под телом сообщения.

- ◆ Нужно ли размещать эти метаданные в содержимом сообщения?

Применяйте собственный тип блока, чтобы вы могли расположить блок в теле сообщения.

- ◆ Могут ли пользователи добавлять несколько копий этих метаданных в сообщение?

Используйте собственный тип блока, чтобы вы могли добавить несколько копий к одному сообщению. Кроме того, вы можете разместить метаполе под сообщением вместе с РНР и кодом JavaScript для добавления дополнительных полей по требованию. Такой вариант рассмотрен в *главе 9*.

Пользовательские классы-оболочки для СРТ

СРТ — это просто посты. Таким образом, вы можете вызвать функцию `get_post($ post_id)`, чтобы получить объект класса `WP_Post` для дальнейшей работы. Для сложных СРТ целесообразно создать класс-оболочку, чтобы вы могли взаимодействовать со своим СРТ более объектно-ориентированным способом.

Основная идея заключается в создании пользовательского класса РНР, который включает в себя в качестве свойства объект публикации, сгенерированный из идентификатора публикации СРТ. В дополнение к хранению этого объекта публикации класс-обертка также содержит методы для всех функций, связанных с этим СРТ.

В листинге 5.4 показан пример класса-оболочки для нашего СРТ `homework`.

```

<?php
/*
    Class wrapper for homework CPT
    /wp-content/plugins/schoolpress/classes/class.homework.php
*/
class Homework {
    // Конструктор может принимать $post_id.
    function __construct($post_id = NULL) {
        if (!empty($post_id))
            $this->getPost($post_id);
    }

    // Получить связанный пост и предварительно заполнить некоторые свойства.
    function getPost($post_id) {
        // получить пост
        $this->post = get_post($post_id);

        // устанавливаем некоторые свойства для легкого доступа
        if (!empty($this->post)) {
            $this->id = $this->post->ID;
            $this->post_id = $this->post->ID;
            $this->title = $this->post->post_title;
            $this->teacher_id = $this->post->post_author;
            $this->content = $this->post->post_content;
            $this->required = $this->post->_schoolpress_homework_is_required;
            $this->due_date = $this->post->due_date;
        }

        // Возвращаем идентификатор записи, если она найдена, или false, если нет.
        if (!empty($this->id))
            return $this->id;
        else
            return false;
    }
}
?>

```

Конструктор этого класса может принять `$post_id` в качестве параметра и передать его методу `getPost()`, который присоединяет объект `$post` к экземпляру класса, а также предварительно заполняет несколько свойств для легкого доступа к ним. В листинге 5.5 показано, как создать экземпляр объекта для конкретного домашнего задания и вывести его содержимое.

```

$assignment_id = 1;
$assignment = new Homework($assignment_id);
echo '<pre>';
print_r($assignment);
echo '</pre>';
//Вывод:
/*
Homework Object
(
    [post] => WP_Post Object
        (
            [ID] => 1
            [post_author] => 1
            [post_date] => 2013-03-28 14:53:56
            [post_date_gmt] => 2013-03-28 14:53:56
            [post_content] => This is the assignment...
            [post_title] => Assignment #1
            [post_excerpt] =>
            [post_status] => publish
            [comment_status] => open
            [ping_status] => open
            [post_password] =>
            [post_name] => assignment-1
            [to_ping] =>
            [pinged] =>
            [post_modified] => 2013-03-28 14:53:56
            [post_modified_gmt] => 2013-03-28 14:53:56
            [post_content_filtered] =>
            [post_parent] => 0
            [guid] => http://schoolpress.me/?p=1
            [menu_order] => 0
            [post_type] => homework
            [post_mime_type] =>
            [comment_count] => 3
            [filter] => raw
            [format_content] =>
        )

    [id] => 1
    [post_id] => 1
    [title] => Assignment 1
    [teacher_id] => 1
    [content] => This is the assignment...
    [required] => 1
    [due_date] => 2013-11-05
)
*/

```


Расширение класса `WP_Post` в сравнении с созданием класса-обертки

Другой вариант — расширить класс `WP_Post`, но в настоящее время это невозможно, поскольку этот класс определен как `final`, т. е. он не может быть расширен. Разработчики WordPress пояснили, что они делают это, чтобы не дать людям создавать плагины, которые основаны на расширении объекта `WP_Post`, поскольку `WP_Post` подлежит пересмотру в будущих версиях WordPress. Мы думаем, что они отстали от жизни³.

В главе 6 мы расширяем класс `WP_User` (который не определен как `final`). Но лучшее, что мы можем сделать с `WP_Post`, — это создать для него класс-обертку.

Зачем нужны классы `Wrapper`?

Создание класса-оболочки для вашего CPT целесообразно по нескольким причинам:

- ◆ Вы можете поместить весь свой код для регистрации CPT в одном месте.
- ◆ Весь свой код для регистрации связанных таксономий можно сосредоточить в одном месте.
- ◆ Вы можете создать все свои функции, связанные с CPT, в качестве методов класса-оболочки.
- ◆ Ваш код будет читаться лучше.

Держите CPT и таксономии вместе

Поместите весь свой код для регистрации CPT и таксономии в одном месте. Вместо того чтобы вводить один блок кода для регистрации CPT и определять таксономии и отдельную оболочку класса для работы с CPT, вы можете поместить определения CPT и таксономии в саму оболочку класса, как показано в следующем примере.

Пример

```
/*
    Обертка класса для homework CPT с функцией init
    /wp-content/plugins/schoolpress/classes/class.homework.php
*/
class Homework
{
    // Конструктор может принимать $post_id.
    function __construct($post_id = NULL)
```

³ А если серьезно, основная команда разработчиков действительно умна и реализует хорошую идею. Если кто-то расширит класс `WP_Post` и создаст новые свойства и методы для своего нового класса, это вызовет проблемы, если в ядре WordPress позже решат использовать свойства и методы в базовом классе `WP_Post` с такими же именами.

```

{
    if(!empty($post_id))
        $this->getPost($post_id);
}

// Получить связанный пост и предварительно заполнить некоторые свойства.
function getPost($post_id)
{
    /* пропуск */
}

// Регистрация CPT и таксономий при init.
function init()
{
    // Зарегистрируем CPT "homework".
    register_post_type(
        'homework',
        array(
            'labels' => array(
                'name' => __('Homework'),
                'singular_name' => __('Homework')
            ),
            'public' => true,
            'has_archive' => true,
        )
    );

    // Регистрация таксономии для предметов.
    register_taxonomy(
        'subject',
        'homework',
        array(
            'label' => __('Subjects'),
            'rewrite' => array('slug' => 'subject'),
            'hierarchical' => true
        )
    );
}
}

// Запускаем инициализацию Homework при выполнении init.
add_action('init', array('Homework', 'init'));

```

Листинг примера был сокращен (полную версию можно найти на сайте GitHub этой книги (github.com/bwawwp)), но в нем показано, как добавить метод `init()` в свой

класс, который подключен к действию инициализации. Затем метод `init()` запускает весь код, необходимый для определения СРТ. Здесь вы также можете определить другие хуки и фильтры с обратными вызовами, связанными с другими методами в классе `Homework`.

Вы можете организовать код иначе, но мы находим, что наличие всего кода, связанного с СРТ, в одном месте очень помогает.

Держите все в классе-обертке

Вы можете создать все свои функции, связанные с СРТ, в качестве методов класса-оболочки. Когда мы зарегистрировали СРТ для домашней работы, на панель инструментов была добавлена страница, позволяющая нам "Редактировать домашнюю работу". Учителя могут создавать домашние задания, как и любой другой пост, с заголовком и содержанием. Учителя могут публиковать домашнее задание, когда оно будет готово для учеников. Все эти функции, связанные с публикациями, доступны бесплатно при создании СРТ.

С другой стороны, большая часть функциональности многих СРТ, включая нашу СРТ домашней работы, должна быть запрограммирована. При наличии класса-оболочки эта функциональность может быть добавлена в качестве методов нашего класса `Homework`.

Например, одно действие, которое мы хотим сделать с нашими домашними заданиями, — это собрать все материалы для определенного задания. Как только мы это сделаем, мы можем отобразить их в виде списка или обработать их каким-либо образом. В листинге 5.6 приведена пара методов, которые мы можем добавить к нашему классу домашних заданий, чтобы собрать связанные материалы и рассчитать шкалу оценок.

Листинг 5.6. Добавление методов в класс `Homework`

```
<?php
class Homework
{
    / * Опущены конструктор и другие методы из предыдущих примеров * /

    /*
        Получить связанные задания.
        Установить для $force значение true,
        чтобы метод снова вызывал дочерние элементы.
    */
    function getSubmissions($force = false)
    {
        // Для этого нужен идентификатор сообщения.
        if (empty($this->id))
            return array();
    }
}
```

```

// Мы уже получили их?
if(!empty($this->submissions) && !$force)
    return $this->submissions;

// Ладно, получайте назначения.
$this->submissions = get_children(array(
    'post_parent' => $this->id,
    'post_type' => 'submissions',
    'post_status' => 'published'
));

// Убедитесь, что назначение является как минимум массивом.
if(empty($this->submissions))
    $this->submissions = array();

return $this->submissions;
}

/*
    Рассчитать кривую оценки

*/
function doFlatCurve($maxscore = 100)
{
    $this->getSubmissions();

    // Определить наивысшую оценку.
    $highscore = 0;
    foreach($this->submissions as $submission)
    {
        $highscore = max($submission->score, $highscore);
    }

    // Вычисляем кривую.
    $curve = $maxscore - $highscore;

    // Исправить более низкие оценки.
    foreach($this->submissions as $submission)
    {
        update_post_meta(
            $submission->ID,
            "score",
            min($maxscore, $submission->score + $curve)
        );
    }
}

}
?>

```

Классы Wrapper читаются лучше

Помимо организации вашего кода для облегчения поиска, работа с классами-оболочками также делает ваш код более легким для чтения и понимания. В следующем примере приведен код программы с полностью упакованными СРТ Homework и Submission и специальными пользовательскими классами (описанными в главе 6).

Пример

```
<?php
// Используем метод класса Student, чтобы проверить,
// является ли текущий пользователь студентом
if (Student::is_student()) {
    // Студент по умолчанию - текущий пользователь.
    $student = new Student();

    // Давайте выясним, когда крайний срок отправить следующее задание
    $assignment = $student->getNextAssignment();

    // Отображение информации и ссылок.
    if (!empty($assignment)) {
        ?>
        <p>Your next assignment
        <a href="<?php echo get_permalink($assignment->id);?>">
        <?php echo $assignment->title;?></a>
        for the
        <a href="<?php echo get_permalink($assignment->class_id);?>">
        <?php echo $assignment->class->title;?></a>
        class is due on <?php echo $assignment->getDueDate();?>.</p>
        <?php
        }
    }
    ?>
```

Программа была бы намного сложнее, если все вызовы метода `get_post()` и циклы по массивам дочерних записей осуществлялись бы напрямую. Объектно-ориентированный подход делает ваш код более доступным для других разработчиков.

Пользователи, их роли и возможности

В *главе 1* мы установили возможность входа как важнейший компонент любого веб-приложения. Одной из замечательных особенностей использования WordPress для ваших приложений является то, что вы получаете полнофункциональное управление пользователями "из коробки". Ядро WordPress включает в себя:

- ◆ Безопасный вход в систему с паролем, который хэшируется с солью.
- ◆ Пользовательские записи с адресом электронной почты, именем пользователя, отображаемым именем, аватаром и биографией.
- ◆ Окно администратора для просмотра, поиска, добавления, редактирования и удаления пользователей.
- ◆ Роли пользователей для отделения администраторов от редакторов, авторов, участников и подписчиков.
- ◆ Страницы для пользователей, чтобы войти, зарегистрироваться и сменить пароль.

Используя различные функции и API WordPress, мы можем сделать следующее:

- ◆ Добавлять пользовательские метаданные или поля профиля для каждого пользователя и управлять ими.
- ◆ Определять пользовательские роли и возможности для более точного контроля над тем, какие пользователи могут получить доступ к каким областям.

Управление пользователями в WordPress — дело довольно простое. Вкладка **User** на панели инструментов позволяет легко просматривать, искать, добавлять, редактировать и удалять пользователей. Также легко управлять пользователями, изменяя программный код.

В этой главе мы рассмотрим следующие вопросы:

- ◆ Как получить доступ к данным пользователя в вашей программе.
- ◆ Как добавить собственные поля для пользователей.
- ◆ Как настроить профили пользователей и отчеты на панели инструментов.
- ◆ Как добавлять, обновлять и удалять пользователей.
- ◆ Как определить пользовательские роли и возможности.
- ◆ *Как расширить пользовательский класс WordPress для создания собственных классов для пользователей.*

Получение данных пользователей

В этом разделе мы рассмотрим, как создать экземпляр объекта пользователя WordPress в программе и как получить основную информацию о пользователе, такую как логин и адрес электронной почты, а также метаданные пользователя из этого объекта.

Класс `WP_User` — "рабочая лошадка" для управления пользователями WordPress через код. Как и все остальное в WordPress и PHP, есть несколько различных способов заставить работать объект `WP_User`. Далее приведены примеры некоторых из самых популярных методов.

Примеры

```
// Получаем объект WP_User, который WordPress создает
// для текущего пользователя, вошедшего в систему
global $current_user;

// получить текущего пользователя, вошедшего в систему,
// с помощью функции wp_get_current_user()
$user = wp_get_current_user();

// установить некоторые переменные
$user_id = 1;
$username = 'jason';
$email = 'jason@strangerstudios.com';

// получить пользователя по ID
$user = wp_get_userdata($user_id);

// получить пользователя по другому полю
$user1 = wp_get_user_by('login', $username);
$user2 = wp_get_user_by('email', $email);

// использовать конструктор WP_User напрямую
$user = new WP_User($user_id);

// используем конструктор WP_User с именем пользователя
$user = new WP_User($username);
```

Если у вас есть объект `WP_User`, то вы можете получить любую часть пользовательских данных, как показано в следующем примере.

Пример

```
// Получить данные только что вошедшего пользователя
$user = wp_get_current_user();
```

```
// вывести имя пользователя
echo $user->display_name;
```

```
// используем адрес электронной почты пользователя для отправки электронного письма
wp_mail($user->user_email, 'Email Subject', 'Email Body');
```

```
// получить любые метаданные пользователя
echo 'Department: ' . $user->department;
```

Вы можете получить доступ к данным, хранящимся в таблице `wp_users` (`user_login`, `user_nicename`, `user_email`, `user_url`, `user_registered`, `user_status` и `display_name`), используя оператор стрелки, например `$user->display_name`.

Получить доступ к любому значению в таблице `wp_usermeta`, также можно посредством оператора стрелки, например `$user->meta_key`, или вызвав функцию `get_user_meta()`. Следующие две строки кода дают одинаковый результат:

```
<?php
$full_name = trim($user->first_name . ' ' . $user->last_name);
$full_name = trim(get_user_meta($user->ID, 'first_name') .
    ' ' . get_user_meta($user->ID, 'last_name'));
?>
```

Полезно понять хитрость, которая реализована в WordPress, чтобы позволить вам обращаться к пользовательским метаданным по требованию, как если бы каждое метаполе было свойством класса `WP_User`. Класс `WP_User` использует переопределенные свойства или "магический метод" `__get()`¹.

С помощью "магических" методов любое свойство объекта `WP_User`, которое вы пытаетесь получить, не являющееся фактическим свойством объекта, будет передано в метод `__get()` класса. Следующий пример иллюстрирует упрощенную² версию метода `__get()`, используемого в классе `WP_User`.

Пример

```
function __get($key) {
    if (isset($this->data->$key)) {
        $value = $this->data->$key;
    } else {
        $value = get_user_meta($this->ID, $key, true);
    }

    return $value;
}
```

¹ Любой метод класса, начинающийся с двух подчеркиваний, считается "магическим" методом в PHP, потому что он "магически" запускается во время определенных событий.

² Для ясности мы изъяли части метода, предназначенные для обратной совместимости и фильтрации в определенных ситуациях. Предыдущий код содержит только существо метода.

Давайте проанализируем этот фрагмент. Сначала метод проверяет, существует ли значение в свойстве `$data` объекта `WP_User`. Если так, то это значение используется. Если нет, метод вызывает функцию `get_user_meta()`, чтобы увидеть, существует ли какое-либо метазначение, через переданный ключ.

Поскольку мы загружаем метазначения по требованию, таким образом, при создании экземпляра нового объекта `WP_User` требуется меньше памяти. С другой стороны, поскольку метазначения недоступны до тех пор, пока вы их не запросите, нельзя получить все метаданные пользователя, указав код вроде `print_r($ user)` или `print_r($ user->data)`.

Чтобы перебрать все метаданные для пользователя, вызовите функцию `get_user_meta()` без переданного ему параметра `$key`:

```
// получить все метаданные для пользователя
$user_meta = get_user_meta($user_id);
foreach($user_meta as $key => $value)
    echo $key . ': ' . $value . '<br />';
```

Знать, как действует функция `__get()` в WordPress, не только интересно, но и важно, потому что вы можете избежать нескольких ограничений этого магического метода.

Методы `__get()` и `__set()` не вызываются, когда задания объединены в цепочку. Например, выражение `$year = $user->graation_year = '2012'` будет давать противоречивые результаты.

Аналогично, `__get()` не вызывается, если находится в вызове функции `empty()` или `isset()`. Так что выражение `if(empty($user->graation_year))` вернет `false`, даже если существуют некоторые пользовательские метаданные с ключом `graduation_year`.

Решить эти две проблемы можно, немного расширив код, как показано в следующем примере.

Пример

```
// Разбиваем назначения на несколько строк при использовании магических методов.
$user->graduation_year = '2012';
$year = '2012';

// Чтобы проверить, является ли метазначение пустым,
// сначала установите локальную переменную.
$year = $user->graduation_year;
if (empty($year))
    $year = '2012';
```

Добавляем, обновляем и удаляем пользователей

Мы уже упоминали некоторые основные функции для добавления, обновления и удаления пользователей в *главе 2*, но поскольку работа с пользовательскими

данными является важной частью любого веб-приложения, здесь приведем краткий обзор с некоторыми дополнительными примерами и разные сценарии использования.

Иногда вам нужно добавлять пользователей программными средствами, а не с помощью панели инструментов WordPress. В нашем приложении SchoolPress мы можем разрешить учителям вводить список адресов электронной почты и генерировать пользователя для каждого введенного адреса электронной почты.

Или, может быть, вы хотите настроить процесс регистрации. Встроенную регистрационную форму WordPress сложно модифицировать. Часто проще создать собственную форму и использовать функции WordPress, чтобы добавить пользователя в бэкенде³.

Как вы уже знаете, функция добавления пользователя в WordPress — это `wp_insert_user()`, которая берет массив пользовательских данных и вставляет их в таблицы `wp_users` и `wp_usermeta`, как показано в следующем примере.

Пример

```
// Вставляем пользователя из собранных нами значений
$user_id = wp_insert_user(array(
    'user_login' => $username,
    'user_pass' => $password,
    'user_email' => $email,
    'first_name' => $firstname,
    'last_name' => $lastname
));
// проверка, если имя пользователя или адрес электронной почты уже использовались
if (is_wp_error($user_id)){
    echo $return->get_error_message();
} else {
    // продолжаем и делаем все, что вы хотите с новым $user_id
}
```

Следующий код обеспечивает автоматический вход в систему после добавления пользовательских данных. Функция `wp_signon()` аутентифицирует пользователя и устанавливает безопасные cookie для входа в систему.

Пример

```
// Ладно, авторизуем их в WP
$creds = array();
$creds['user_login'] = $username;
```

³ Вот так плагин Paid Memberships Pro регистрирует пользователей со страницы оформления заказа.

```
$creds['user_password'] = $password;
$creds['remember'] = true;
$user = wp_signon($creds, false);
```

Обновлять пользователей так же просто, как добавлять их, с помощью функции `wp_update_user()`. Вы передаете массив пользовательских данных и метаданных. Пока в массиве есть ключ `ID` с допустимым идентификатором пользователя в качестве значения, WordPress будет устанавливать любые указанные пользователем значения.

Пример

```
// Это обновит электронную почту пользователя и оставит другие значения в покое
$userdata = array('ID' => 1, 'user_email' => 'jason@strangerstudios.com');

wp_update_user($userdata);

// эта функция также идеально подходит для одновременного
// обновления нескольких пользовательских метаполей
wp_update_user(array(
    'ID' => 1,
    'company' => 'Stranger Studios',
    'title' => 'CEO',
    'personality' => 'crescent fresh'
));
```



Вы не можете обновить `user_login` пользователя через `wp_update_user`. Кроме того, если `user_pass` пользователя обновлен, то будет осуществлен выход пользователя из системы. Вы можете использовать предыдущий пример автологина, чтобы снова войти в систему с новым паролем.

Функция `update_user_meta($user_id, $meta_key, $meta_value, $prev_value)` позволяет обновлять одно метазначение пользователя одновременно.

Следующие примеры иллюстрируют некоторые дополнительные функции.

Примеры

```
// Массивы будут сериализованы
$children = array('Isaac', 'Marin');
update_user_meta($user_id, 'children', $children);

// вы также можете сохранить массив, сохранив несколько
// значений с одним и тем же ключом
add_user_meta($user_id, 'children', 'Isaac');
add_user_meta($user_id, 'children', 'Marin');

// при сохранении нескольких значений указываем параметр $prev_value, чтобы
выбрать, какой из них изменить
```

```
update_user_meta($user_id, 'children', 'Isaac Ford', 'Isaac');
update_user_meta($user_id, 'children', 'Marin Josephine', 'Marin');
```

```
// удаляем все пользовательские метаданные по ключу
delete_user_meta($user_id, 'children');
```

```
// удаляем только одну строку при наличии нескольких значений для одного ключа
delete_user_meta($user_id, 'children', 'Isaac Ford');
```

Обратите внимание, что в этом примере мы показываем два разных способа хранения массивов в пользовательских метаданных. Это похоже на сохранение параметров с помощью метода `update_option()` или метаданных публикации с помощью метода `update_post_meta()`. Первый метод (одно сериализованное значение на ключ) ведет обратный отсчет строк в таблице `wp_usermeta`, что позволяет быстрее выполнять запросы с помощью `meta_key`.

Второй метод (несколько значений на ключ) позволяет выполнять запросы по метазначению. Например, хранение дочерних имен в виде отдельных метазаписей пользователя позволяет выполнять такие запросы:

```
<?php
// получаем идентификаторы всех пользователей, у кого есть дети с именем Исаак
$parents_of_isaac = $wpdb->get_col("SELECT user_id
    FROM $wpdb->usermeta
    WHERE meta_key = 'children'
    AND meta_value = 'Isaac'");
?>
```

Хотя можно запрашивать таблицы `wp_usermeta` и `wp_postmeta` по значению `meta_value`, но при этом следует учитывать время выполнения запроса. Столбец `meta_value` не индексируется, поэтому запросы к большим наборам данных могут быть медленными. Подобные отношения "многие-к-одному" также могут быть сохранены в пользовательских таксономиях, которые обеспечат лучшую производительность.

Удалить пользователя в коде, хотя это и опасно, невероятно легко, что иллюстрирует следующий пример.

Пример

```
// Данный файл содержит wp_delete_user и не всегда загружается,
// поэтому давайте удостоверимся в этом
require_once(ABSPATH . '/wp-admin/includes/user.php');
```

```
// удаляем пользователя
wp_delete_user($user_id);
```

```
// или удалить пользователя и переназначить его сообщения пользователю с ID #1
wp_delete_user($user_id, 1);
```

Для сетевого сайта вам потребуется функция `wpmu_delete_user()`, чтобы удалить пользователя из всей сети. В противном случае `wp_delete_user()` просто удаляет пользователя из текущего блога. Вы можете вызвать функцию `is_multisite()`, чтобы определить, какую функцию следует использовать:

```
// Я хочу убедиться, что мы действительно удалили пользователя.
if (is_multisite())
    wp_delete_user($user_id);
else
    wpmu_delete_user($user_id);
```

Хуки и фильтры

Возможно, более распространенным, чем самостоятельное добавление и обновление пользовательских данных, являются сценарии, в которых вы хотите выполнить какой-то другой фрагмент кода при добавлении или удалении новых пользователей. Например, вы можете захотеть создать и связать новую запись CPT с пользователем при регистрации. Или, может быть, вы хотите очистить соединения и данные, хранящиеся в пользовательских таблицах, когда пользователь удаляется. Вы можете сделать это через некоторые пользовательские хуки и фильтры.

Хук для запуска кода после регистрации пользователя — `user_register`. В следующем примере хук передает идентификатор вновь созданного пользователя.

Пример

```
// Создаем новый CPT "course" (курс), когда учитель регистрируется
function sp_user_register($user_id) {
    // проверяем, является ли новый пользователь учителем (подробнее см. главу 15)
    if (pmpo_hasMembershipLevel('teacher', $user_id)) {
        // добавить новый CPT "курс" с этим пользователем в качестве автора
        wp_insert_post(array(
            'post_title' => 'My First Course',
            'post_content' => 'This is a sample course...',
            'post_author' => $user_id,
            'post_status' => 'draft',
            'post_type' => 'course'
        ));
    }
}

add_action('user_register', 'sp_user_register');
```

Хук для запуска кода до удаления пользователя — `delete_user`. Похожий хук `deleted_user` (обратите внимание на прошедшее время) запускается только *после* удаления пользователя.

Эти хуки в основном взаимозаменяемы, но есть несколько моментов, на которые следует обратить внимание:

- ◆ Если вызвать хук `delete_user` достаточно рано, то можно прервать процесс удаления пользователя.
- ◆ В случае `deleted_user` нужно учесть, что некоторые пользовательские данные и соединения могут быть уже удалены и недоступны.

Пример

```
<?php
// Отправляем электронное письмо, когда пользователь удаляется
function sp_delete_user($user_id){
    $user = get_userdata($user_id);
    wp_mail($user->user_email,
        "You've been deleted.",
        'Your account at SchoolPress has been deleted.'
    );
}
// необходимо получить user_email, поэтому подключаемся раньше
add_action('delete_user', 'sp_delete_user');
?>
```

Что такое роли и возможности?

Роли и возможности — это инструменты WordPress для контроля за доступом пользователя при просмотре и работе на вашем сайте. Каждому пользователю можно назначить одну роль, и любая роль может иметь одну или несколько возможностей. Каждая возможность определяет, может ли пользователь просматривать определенный тип содержимого или выполнять определенное действие.

В каждой установке WordPress есть пять ролей по умолчанию: администратор, редактор, автор, участник и подписчик. Если вы работаете с сетевым сайтом, то будет шестая роль, супер администратор (Super Administrator), которая обеспечивает доступ администратора ко всем сайтам в сети. Полный список возможностей и их соответствие стандартным ролям WordPress можно найти на странице "Роли и возможности, поддерживаемые WordPress" (oreil.ly/jEFUe).

Вскоре мы рассмотрим, как создавать новые роли вне настроек WordPress по умолчанию. Однако для большинства приложений имеет смысл придерживаться ролей по умолчанию: пусть администраторы вашего приложения имеют роль администратора, а все ваши пользователи/клиенты — роль подписчика.

Если пользователи вашего приложения будут создавать контент, подумайте о том, чтобы сделать их авторами (они могут создавать и публиковать посты) или участниками (могут создавать, но не публиковать сообщения). Если в вашем приложении есть модераторы, сделайте их редакторами.

Преимущество ролей по умолчанию — удобство при взаимодействии с плагинами, которые ожидают, что ваши пользователи будут иметь одну из этих ролей. Если ваши администраторы на самом деле являются пользователями в роли менеджера офиса, то вам, возможно, потребуется установить сторонний плагин для работы с такими пользователями. Но иногда бывает и наоборот. Может потребоваться скрыть функциональность, доступную вашим пользователям, в зависимости от их ролей, особенно если у вас есть роли вне администратора (могут иметь доступ ко всему) и подписчиков (могут только просматривать материалы).

Проверка роли и возможностей пользователя

Время от времени вам нужно проверять, обладает ли пользователь соответствующими правами, прежде чем позволить ему сделать что-либо. Это можно реализовать с помощью функции `current_user_can()`, которая принимает один параметр `$capability`, который является строковым значением для возможности. Следующий пример кода иллюстрирует использование этой функции.

Пример

```
if (current_user_can('manage_options')) {  
    // имеет возможность управления параметрами, обычно администратором  
}  
  
if (current_user_can('edit_user', $user_id)) {  
    // можем редактировать пользователя с ID = $user_id.  
    // обычно либо сам пользователь, либо админ  
}  
  
if (current_user_can('edit_post', $post_id)) {  
    // текущий пользователь может редактировать сообщение с ID = $post_id  
    // обычно автор поста или администратор или редактор  
}  
  
if (current_user_can('subscriber')) {  
    // один из способов проверить, является ли текущий пользователь подписчиком  
}
```

Функция `user_can()` позволяет проверить, есть ли некоторая возможность у кого-то, кроме текущего пользователя. В качестве аргументов передайте `$user_id` пользователя, которого вы хотите проверить, возможности и любые другие необходимые аргументы.

```

<?php
/*
    Вывод комментариев для текущего поста,
    подсвечивание любого, кто имеет возможность редактировать их.
*/
global $post; // текущая публикация, на которую мы смотрим

$comments = get_comments('post_id=' . $post->ID);
foreach($comments as $comment) {
    // классы CSS по умолчанию для всех комментариев
    $classes = 'comment';

    // добавляем can-edit CSS класс для авторов
    if (user_can($comment->user_id, 'edit_post', $post->ID)) {
        $classes .= ' can-edit';
    }
}
?>

<div id="comment-<?php echo $comment->comment_ID;?>"
    class="<?php echo $classes;?>">
    Comment by <?php echo $comment->comment_author; ?>:
    <?php echo wpautop($comment->comment_content); ?>
</div>
<?php
}

```

И хотя с помощью функции `current_user_can()` можно проверить роль пользователя, лучше проверять возможности пользователя, а не роли. Это позволяет вашей программе продолжать работать, даже если пользователям назначены разные роли, или ролям назначены разные возможности. Например, проверка для `manage_options` будет работать, если вы укажете, является ли пользователь администратором или пользователем с добавленной возможностью `manage_options`.

Проверка роли пользователя должна быть ограничена случаями, когда вам действительно нужно знать роль пользователя, а не возможность. Если вы обнаружите, что проверяете чью-то роль перед выполнением определенных действий, то должны воспринимать это как подсказку, что вам нужно добавить новую возможность.

В следующем примере приведена функция для обновления любого подписчика, идентификатор которого передается на назначение роли автора. Для большей уверенности мы проверяем массив ролей объекта пользователя вместо вызова функции `user_can()`. Мы применяем метод `set_role()` пользовательского класса, чтобы установить новую роль.

Пример

```
function upgradeSubscriberToAuthor($user_id) {
    $user = new WP_User($user_id);
    if (in_array('subscriber', $user->roles)) {
        $user->set_role('author');
    }
}
```

Создание собственных ролей и возможностей

Как мы уже говорили ранее, желательно придерживаться стандартных ролей WordPress, если это возможно. Однако, если у вас есть разные классы пользователей и вам нужно по-новому разграничить то, что они могут, добавьте пользовательские роли и возможности.

В нашем приложении SchoolPress учителя — авторы, а студенты — подписчики. Однако нам нужна настраиваемая роль для офис-менеджеров, которые могут управлять пользователями, но не могут редактировать контент, темы, параметры, плагины или общие настройки WordPress. В следующем примере показано, как можно настроить роль Office Manager.

Пример

```
function sp_roles_and_caps() {
    // Роль офис-менеджера
    remove_role('office'); // в случае, если мы обновили возможности ниже
    add_role('office', 'Office Manager', array(
        'read' => true,
        'create_users' => true,
        'delete_users' => true,
        'edit_users' => true,
        'list_users' => true,
        'promote_users' => true,
        'remove_users' => true,
        'office_report' => true // новый заголовок нашего пользовательского отчета
    ));
}
// запускаем эту функцию при активации плагина
register_activation_hook(__FILE__, 'sp_roles_and_caps');
```

При вызове функции `add_role()` она обновляет параметр `wp_user_roles` в таблице `wp_options`, где WordPress ищет информацию о ролях и возможностях. Таким образом, нужно запускать эту функцию только один раз при активации, а не каждый раз во время выполнения. Вот почему мы регистрируем эту функцию с помощью вызова `register_activation_hook()`.

Мы также выполняем функцию `remove_role('office')` в начале, что важно, если вы хотите полностью удалить роль, но также полезно, чтобы очистить роль "office" перед ее добавлением в случае, если вы отредактировали возможности или другие настройки для роли. Без строки `remove_role()` строка `add_role()` не запустится, поскольку роль уже существует.

Функция `add_role()` принимает три параметра: имя роли, отображаемое имя и массив возможностей. На сайте поддержки WordPress (wordpress.org/support/) вы можете найти имена возможностей по умолчанию или отыскать их в файле `/wp-admin/includes/schema.php` вашей инсталляции WordPress.



WordPress выполняет хук активации только тогда, когда активизируется плагин. Он не будет работать при обновлении плагина или когда вы просто измените один из файлов PHP в плагине. Чтобы получить новые роли и возможности пользователей, вам может понадобиться деактивировать и повторно активировать плагин. Если вы создаете общедоступный плагин, то можете проверить параметры роли и возможностей в хуке `admin_init` и сбросить роли и возможности, если это необходимо. Создатели плагина **Paid Memberships Pro** опубликовали на GitHub файл ([orell.ly/20f01](https://github.com/orell/20f01)), демонстрирующий, как это делается.

Добавить новые возможности пользователя так же просто, как включить имя новой возможности в вызов `add_role()` или использовать метод `add_cap()` в существующей роли. Вот пример, показывающий, как получить экземпляр класса роли и добавить к нему возможность:

```
// даем администраторам office_report, чтобы они могли просматривать этот отчет
$role = get_role('administrator');
$role->add_cap('office_report');
```

Опять же, этот код необходимо запустить только один раз, что сохранит его в базе данных. Поместите подобный код в функцию, зарегистрированную через `register_activation_hook()`, как в последнем примере.

Вы также можете задействовать метод `remove_cap()` класса ролей, что полезно, если вы хотите удалить некоторые функции из ролей по умолчанию. Например, следующий код удалит возможности `edit_pages` у редакторов, чтобы они могли редактировать любой пост блога, но не страницу (сообщение типа "страница"):

```
// не позволяем редакторам редактировать страницы
$role = get_role('editor');
$role->remove_cap('edit_pages');
```

Добавляя и редактируя роли и возможности, вы получаете мощный инструмент для работы. Определение того, какие пользователи имеют доступ к просмотру и что делают, является важной частью создания приложения. Различные роли могут быть созданы для разных уровней членства или обновлений, связанных с вашим продуктом. В *главе 15* представлен плагин **Paid Memberships Pro**, который добавляет "уровни членства" в качестве отдельной классификации для ваших пользователей. Это может заменить пользовательские роли, но чаще используется вместе с ними. Подробнее о том, как уровни членства и роли могут работать вместе, рассказывается в *главе 15*.

Расширение класса *WP_User*

Подобно тому, как мы обернули класс `WP_Post` в *главе 5*, чтобы создать более конкретный класс для наших пользовательских типов записей, мы можем расширить класс `WP_User` для создания полезных классов, которые помогут нам организовать код, связанный с различными типами пользователей.

Например, в нашем приложении *SchoolPress* есть два основных типа пользователей: `Teacher` и `Student`. В основе обоих типов лежат только пользователи *WordPress*, но у каждого типа пользователей также есть уникальная функциональность. Мы можем инкапсулировать эту уникальную функциональность, написав классы `Teacher` и `Student`, которые расширяют класс `WP_User`.

Разве не здорово, если бы мы могли написать такой код?

Пример

```
<?php
// Student - это класс, расширяющий WP_User
$student = new Student();
foreach($student->getAssignments() as $assignment) {
    // присваивание здесь является экземпляром класса, который расширяет
    WP_Post
    $assignment->print();
}
?>

// тот же код, написанный в менее объектно-ориентированном стиле
$student = wp_get_current_user(); // возврат объекта WP_User для текущего
пользователя
foreach(sp_getAssignmentsByUser($student->ID) as $assignment) {
    sp_printAssignment($assignment->ID);
}
```

Оба блока кода функционально эквивалентны, но первый фрагмент легче читать и работать с ним тоже удобнее. Возможно, еще важнее, чтобы все ваши функции, связанные со студентами, были закодированы как методы в классе `Student`, что поможет вам все упорядочить.

Следующий пример иллюстрирует метод инициализации и `getAssignments()` для класса `Student`.

Пример

```
<?php
class Student extends WP_User {
    // нет конструктора, поэтому используется конструктор WP_User
```

```

// метод для получения заданий для пользователя Student
function getAssignments() {
    // получаем задания через get_posts, если их у нас еще нет
    if (! isset($this->data->assignments))
        $this->data->assignments = get_posts(array(
            'post_type' => 'assignment', // задания
            'numberposts' => -1, // все публикации
            'author' => $this->ID // идентификатор пользователя для этого
ученика
        ));

    return $this->data->assignments;
}

// магический метод для определения $student-> assignments
function __get($key) {
    if ($key == 'assignments') {
        return $this->getAssignments();
    } else {
        // возврат к стандартному магическому методу WP_User
        return parent::__get($key);
    }
}
}
?>

```

Здесь мы определяем класс `Student` для расширения класса `WP_User`, просто добавляя `extends WP_User` к определению класса.

Мы не пишем нашу собственную функцию конструктора, потому что хотим использовать ту же самую, что и класс `WP_User`. А именно, мы хотим получить идентификатор ученика/пользователя, чтобы можно было написать выражение `$student = new Student ($ user_ID);`.

Метод `getAssignments()` использует функцию `get_posts()` для получения всех сообщений типа "задание", назначенных этому студенту. Мы храним массив сообщений о назначениях в свойстве `$data`, которое определено в классе `WP_User` и хранит все основные пользовательские данные и метаданные. Это позволяет нам с помощью выражения вроде `$student->assignments` получить задания позже.

Обычно, если `$student->assignments` является определенным свойством `$student`, будет возвращено значение этого свойства. Но если нет свойства "задания", PHP будет отправлять `assignments` ("задания") в качестве параметра `$key` в ваш метод `__get`. Здесь мы проверяем, что `$key == "assignments"` и затем возвращаем значение метода `getAssignments()`, определенного позже. Если `$key` является чем-то отличным от "assignments", мы передаем это значение в метод `__get()` родительского класса `WP_User`, который проверяет значение в свойстве `$data` экземпляра класса или, в противном случае, отправляет ключ в функцию `get_user_meta()`.

На первый взгляд, здесь всего лишь написано `$student->assignments` вместо `$student->getAssignments()`, что технически верно. Тем не менее такой способ кодирования позволяет нам кэшировать назначенные задания в свойстве `$data` объекта, поэтому нам не нужно запрашивать их снова, если к ним обращаются более одного раза. В результате ваш код окажется более совместимым с другим кодом WordPress: если вы хотите получить электронную почту студента, это будет `$student->user_email`; если вам нужно имя студента, — `$student->first_name`; если вам нужны задания студента, — `$student->assignments`. Человек, использующий вашу программу, не должен знать, что одно из значений хранится в таблице `wp_users`, другое — в `wp_usermeta`, а третье является результатом запроса постов.

Добавление полей регистрации и профиля

Очень часто в вашем приложении нужно будет добавлять дополнительные поля профиля для пользователей. В предыдущем разделе мы обсуждали, как использовать функции `wp_update_user()` и `update_user_meta()` для управления этими значениями. Теперь рассмотрим, как добавить редактируемые поля для наших пользовательских метаданных на страницы регистрации и профиля.

В приложении SchoolPress нам нужно собрать некоторые данные о наших пользователях. Для студентов мы хотим указать их год выпуска, основную специализацию, вторую специализацию и имя консультанта. Для учителей мы хотим знать их отдел и местонахождение офиса. Для обоих типов пользователей мы хотим указать их пол, возраст и номер телефона.

Есть несколько различных плагинов, которые помогут вам сделать это быстрее. Например, если вы установите плагин Paid Memberships Pro Register Helper⁴, то сможете использовать код из листинга 6.1, чтобы добавить эти поля на страницы регистрации и профиля.

Листинг 6.1. Регистрация дополнительных полей для пользователей

```
<?php
function ps_registration_fields(){
    // сохраняем поля в массиве
    $fields = array();

    // поля для всех пользователей
    $fields[] = new PMProRH_Field(
        'gender',
        'select',
        array(
```

⁴ Paid Memberships Pro Register Helper (oreil.ly/DD15) был создан для работы с Paid Memberships Pro, но будет работать и без него.

```

        'options' => array(
            '' => 'Choose One',
            'male' => 'Male',
            'female' => 'Female',
            'other' => 'Other',
            'undisclosed' => 'Prefer not to say'
        ),
        'profile' => true,
        'required' => true
    )
);
$fields[] = new PMProRH_Field(
    'age',
    'text',
    array(
        'size' => 10,
        'profile' => true,
        'required' => true
    )
);
$fields[] = new PMProRH_Field(
    'phone',
    'text',
    array(
        'size' => 20,
        'label' => 'Phone Number',
        'profile' => true,
        'required' => true
    )
);

// поля для учителей
$fields[] = new PMProRH_Field(
    'department',
    'text',
    array(
        'size' => 40,
        'profile' => true,
        'required' => true
    )
);
$fields[] = new PMProRH_Field(
    'office',
    'text',
    array(
        'size' => 40,
        'profile' => true,

```

```

        'required' => true
    )
);

// поля для студентов
$fields[] = new PMProRH_Field(
    'graduation_year',
    'text',
    array(
        'label' => 'Expected Graduation year',
        'size' => 10,
        'profile' => true,
        'required' => true
    )
);

$fields[] = new PMProRH_Field(
    'major',
    'text',
    array('size' => 40, 'profile' => true, 'required' => true)
);

$fields[] = new PMProRH_Field(
    'minor',
    'text',
    array('size' => 40, 'profile' => true)
);

// добавляем поля на страницу регистрации
foreach($fields as $field) {
    pmporh_add_registration_field('after_password', $field);
}

}

add_action('init', 'ps_registration_fields');
?>

```

Вы можете найти подробные инструкции по использованию PMPro Register Helper и синтаксис для определения полей в файле плагина *README*. Вместо того чтобы освещать это здесь, давайте рассмотрим подробнее добавление одного поля к страницам регистра и профиля вручную, используя те же хуки и фильтры, которые делает PMPro Register Helper. В следующем примере показано, как добавить наше поле на страницу регистрации.

Пример

```

<?php
function sp_register_form(){
    // получаем значение возраста, переданное в форму
    if (! empty($_REQUEST['age']))

```

```

        $age = intval($_REQUEST['age']);
    else
        $age = '';

    // показать ввод
    ?>
    <p>
    <label for="age">Age<br />
    <input type="text" name="age" id="age" class="input"
        value="<?php echo esc_attr($age); ?>" />
    </label>
    </p>
    <?php
}
add_action('register_form', 'sp_register_form');
?>

```



Мы проверяем условие `if(! empty($_REQUEST['age']))`, чтобы избежать предупреждения PHP, когда пользователи впервые посещают страницу регистрации, и в переменной `$REQUEST` еще нет данных формы. Мы также используем функцию `esc_attr()` при выводе возраста в значение атрибута поля ввода. Функции обработки ввода описаны подробно в *главе 8*.

В следующем примере мы обновляем возраст нашего пользователя после регистрации.

Пример

```

function sp_register_user($user_id){
    // получаем значение возраста, переданное в форму
    $age = intval($_REQUEST['age']);

    // обновляем метаданные пользователя
    update_user_meta($user_id, 'age', $age);
}
add_action('register_user', 'sp_register_user');

```

В следующем примере мы добавляем поле возраста на страницу профиля пользователя. Нам нужно подключиться как к `show_user_profile`, так и к `edit_user_profile`, чтобы показать наше настраиваемое поле, когда пользователи просматривают свой собственный профиль и когда администраторы редактируют профили других пользователей.

Пример

```

<?php
function sp_user_profile($user){
    // показать ввод

```



```

    $age = $user->age;
    ?>
    <table class="form-table">
    <tbody>
    <tr>
        <th><label for="age">Age</label></th>
        <td>
            <input type="text" name="age" id="age" class="input"
            value="<?php echo esc_attr($age); ?>" />
        </td>
    </tr>
    </tbody>
    </table>
    <?php
}
// собственный профиль пользователя
add_action('show_user_profile', 'sp_user_profile');
// администраторы редактируют профили пользователей
add_action('edit_user_profile', 'sp_user_profile');
?>

```

Обратите внимание, что по умолчанию HTML-теги `<p>` на странице регистрации WordPress служат для разделения полей, в то время как HTML-профиль профиля по умолчанию на панели инструментов использует строки таблицы.

В следующем примере мы обновляем наше поле при обновлении профиля.

Пример

```

<?php
function sp_profile_update($user_id){
    // убедитесь, что текущий пользователь может
    // редактировать данные этого пользователя
    if (! current_user_can('edit_user', $user_id)) {
        return false;
    }

    // проверка, было ли задано значение
    if (isset($_POST['age'])){
        // обновляем метаданные пользователя
        update_user_meta($user_id, 'age', intval($_POST['age']));
    }
}
// собственный профиль пользователя
add_action('personal_options_update', 'sp_profile_update');
// редактирование администратором
add_action('edit_user_profile_update', 'sp_profile_update');
?>

```

Опять же, мы подключаемся к двум отдельным хукам: один, когда пользователи просматривают свой собственный профиль, и другой, когда администраторы редактируют профили других пользователей.

Вот так вы можете добавить поле на страницу регистрации и на страницу профиля. Просто повторите это для каждого поля, которое вы хотите добавить (или установите плагин, такой как PMPro Register Helper, чтобы он сделал это за вас), и все готово.

Настройка таблицы пользователей на административной панели

Учитывая необходимость дополнительных метаданных для наших пользователей, иногда приходится расширять основную таблицу пользователей на панели инструментов WordPress.

Вы можете создать свою собственную страницу администратора с пользовательскими запросами и отчетом, который имитирует стиль таблиц панели мониторинга (как мы это сделали для "Списка участников" в Paid Memberships Pro). Или можно применить хуки и фильтры, предоставляемые WordPress, чтобы добавить столбцы и фильтры к стандартному списку пользователей, что мы и рассмотрим сейчас.

В следующем примере мы используем фильтры `manage_users_columns` и `manage_users_custom_column`. Давайте добавим в список пользователей поле возраста.

Пример

```
// Добавляем наш столбец в таблицу
function sp_manage_users_columns($columns){
    $columns['age'] = 'Age';
    return $columns;
}
add_filter('manage_users_columns', 'sp_manage_users_columns');

// сообщаем WordPress, как заполнять столбец
function sp_manage_users_custom_column($value, $column_name, $user_id){
    $user = get_userdata($user_id);
    if ($column_name == 'age')
        $value = $user->age;

    return $value;
}
add_filter('manage_users_custom_column',
    'sp_manage_users_custom_column', 10, 3);
```

Фильтр `manage_users_columns` передает массив, содержащий все столбцы WordPress по умолчанию (и любые, добавленные другими плагинами). Вы можете добавить

столбцы, удалить их (используя `unset($columns['column_name'])`) и изменить их порядок. Ключи в массиве `$columns` являются уникальными строками для их идентификации. Значения в массиве `$columns` — это заголовки для каждого столбца.

Фильтр `manage_users_custom_column` применяется к каждому столбцу в массиве `manage_users_columns`, который не является столбцом WordPress по умолчанию (т. е. это любой пользовательский столбец, который вы добавляете). В обратном вызове `sp_manage_users_custom_column()` вы можете выполнить любые вычисления, необходимые для получения значений для каждого настраиваемого столбца. Как правило, функция содержит большой блок `if/then/else` или оператор `switch`, проверяющий значение `$column_name` и возвращающий правильное значение для каждого столбца.

Если вы выполните код предыдущего примера, то получите столбец **Age**, добавленный на страницу ваших пользователей, но по умолчанию вы не сможете щелкнуть по нему, чтобы отсортировать список пользователей по возрасту, как вы можете с некоторыми из столбцов по умолчанию списка пользователей. Следующий пример показывает, как это реализовать.

Пример

```
<?php
// сделать столбец сортируемым
function sp_manage_users_sortable_columns($columns){
    $columns['age'] = 'Age';
    return $columns;
}
add_filter('manage_users_sortable_columns',
    'sp_manage_users_sortable_columns');
// обновляем user_query при сортировке по возрасту функции
function sp_pre_user_query($user_query){
    global $wpdb, $current_screen;
    // убедитесь, что мы просматриваем список пользователей на панели инструментов
    if ($current_screen->id != 'users') {
        return;
    }
    // упорядочить по возрасту
    if ($user_query->query_vars['orderby'] == 'Age') {
        $user_query->query_from .= " INNER JOIN $wpdb->usermeta m1
ON $wpdb->users u1
AND (u1.ID = m1.user_id)
AND (m1.meta_key = 'age')";
        $user_query->query_orderby = " ORDER BY m1.meta_value
" . esc_sql($user_query->query_vars['order']);
    }
}
add_action('pre_user_query', 'sp_pre_user_query');
?>
```

Здесь мы определили возраст как сортируемый столбец с помощью фильтра `manage_users_sortable_columns`. С помощью фильтра `pre_user_query` мы обнаруживаем параметр `&sortby=Age` на странице списка пользователей и обновляем объект `$user_query` для запроса к таблице `wp_usermeta` и упорядочения по возрасту. Обратите внимание, как мы используем глобальную переменную `$current_screen`, установленную на странице администратора, чтобы убедиться, что мы находимся на странице списка пользователей перед редактированием запроса.

Значение `$user_query->query_vars[order]` будет `ASC` или `DESC`, но мы обернем переменную в функции `esc_sql()`, чтобы защитить наш запрос от SQL-инъекций. Еще лучше было бы проверить, что значение является точно `ASC` или `DESC`, и выдать ошибку для любого другого значения.

Плагины

Теперь, когда вы увидели, как настраивать различные аспекты системы управления пользователями WordPress, давайте рассмотрим несколько пользовательских плагинов, которые сделают вашу жизнь в веб-приложениях немного проще.

Theme My Login

Вашим участникам не нужно знать, что ваш сайт построен на WordPress. Вы можете использовать форму входа, которая легко интегрируется с дизайном вашего сайта, а не вход в WordPress по умолчанию. Плагин Theme My Login (bwawwp.com/theme-my-login) прекрасно справляется с этой задачей. Трафик к файлу `wp-login.php` перенаправляется на страницу входа, которая выглядит как остальная часть вашего сайта, а не бэкэнд WordPress.

В плагине Theme My Login также есть полезные платные модули для настройки тем профилей пользователей, сокрытия панели мониторинга от пользователей, не являющихся администраторами, и контроля перенаправления пользователей при входе и выходе из системы.

Hide the Admin Bar

Плагин Hide the Admin Bar (bit.ly/hide-bar) в точности соответствует названию. Только администраторы увидят панель администратора WordPress при просмотре веб-интерфейса вашего сайта. Плагин представляет собой всего несколько строк кода и может быть отредактирован в соответствии с вашими потребностями, например так, чтобы редакторы и авторы могли просматривать панель администратора.

Paid Memberships Pro

Плагин Paid Memberships Pro (bwawwp.com/paid-pro) разработан Stranger Studios и позволяет вам монетизировать контент на вашем сайте, создав сообщество с подписками. Он идеально подходит для любого бизнеса или блофера, желающего за-

блокировать часть или весь контент или взимать плату за предоставленные услуги. Этот плагин легко интегрируется с платежными шлюзами, такими как Stripe, PayPal и Authorize.net. Настройки для периодических или разовых платежей включены. Paid Memberships Pro позволяет создавать различные уровни членства на вашем сайте.

PMPro Register Helper

Плагин PMPro Register Helper (bit.ly/pmp-reg-help) изначально был написан для работы с Paid Memberships Pro, но подойдет и для других проектов. Этот плагин упрощает процесс добавления дополнительных полей в формы регистрации и профиля. Вместо набора из трех хуков и функций для каждого поля, поля могут быть добавлены в виде пары строк кода, как показано в следующем примере.

Пример

```
<?php
$text = new PMProRH_Field(
    'company',
    'text',
    array(
        'size' => 40,
        'class' => 'company',
        'profile' => true,
        'required' => true
    )
);
pmporh_add_registration_field('after_billing_fields', $text);
?>
```

Плагин Register Helper также имеет шорткоды для вставки форм регистрации на ваши страницы, а также боковые панели и модули, которые служат отправными точками для ваших собственных страниц регистрации, профиля и страниц каталога участников.

Members

Плагин Members (<http://bwawwp.com/members-wp>) расширяет контроль над ролями и возможностями пользователей на вашем сайте. Это позволяет вам редактировать, а также создавать и удалять пользовательские роли и права. Этот плагин также позволяет вам устанавливать разрешения для различных пользовательских ролей, чтобы определить, какие роли имеют возможность добавлять, редактировать и/или удалять различные фрагменты содержимого.

Вы всегда можете написать свой собственный код для добавления ролей и возможностей, но Members обеспечивает приятный графический интерфейс поверх этой функциональности, который часто бывает полезен.

WP User Fields

Это шутка. Еще нет плагина под названием "WP User Fields". До сих пор не существует хорошего, удобного для пользователя плагина для добавления пользовательских полей на стандартную страницу регистрации WordPress, профилей и списка пользователей. Лучшие решения основаны на других плагинах для форм.

Может быть вы как раз и создадите плагин для пользовательских полей, который нужен WordPress? Закончите читать эту книгу, и у вас будут инструменты для этого. Начните со следующей главы, посвященной API-интерфейсам в WordPress, объектам и вспомогательным функциям.

Работа с API-интерфейсами WordPress, объектами и вспомогательными функциями

В этой главе мы обратим внимание на несколько API, объектов и вспомогательных функций WordPress, которые не рассматриваются в других главах книги, но все еще являются важными частями арсенала разработчика WordPress.

API шорткодов

Шорткоды — это специально отформатированные фрагменты текста, с помощью которых можно вставлять динамические элементы в ваши публикации, страницы, виджеты и другие области статического содержимого.

Шорткоды бывают трех основных видов:

1. Единичный шорткод вроде `[myshortcode]`
2. Шорткоды с атрибутами, как `[myshortcode id = "1" type = "text"]`
3. Заключающие шорткоды, например так `[myshortcode id = "1"] ... некоторый контент здесь ... [/ myshortcode]`

В *главе 3* мы поделились примером того, как использовать шорткоды для добавления произвольного контента в пост или страницу WordPress. В примере, рассмотренном далее, как и раньше, мы просто заменили шорткод на наше содержимое. Вы также можете добавить атрибуты к шорткоду, чтобы повлиять на функцию обратного вызова, обрабатывающую его, или обернуть некоторый контент в пару открывающих и закрывающих шорткодов, чтобы отфильтровать некоторый конкретный контент.

Основной этап создания шорткодов — определение функции обратного вызова для вашего шорткода с помощью функции `add_shortcode()`. Любые атрибуты добавляются в массив, который передается обратному вызову в качестве первого параметра `$atts`. Произвольное вложенное содержимое передается обратному вызову в качестве второго параметра — `$content`.

Следующий пример создает шорткод с именем `msg` и задействует атрибуты и вложенный контент.

```

<?php
/*
    Обратный вызов для шорткода [msg]
    Пример: [msg type = "error"] Это сообщение об ошибке. [/msg]
    Вывод:
    <div class="message message-error">
        <p> Это сообщение об ошибке. </p>
    </div>
*/
function sp_msg_shortcode($atts, $content)
{
    // атрибуты по умолчанию
    extract(shortcode_atts(array(
        'type' => 'information',
    ), $atts));
    $content = do_shortcode($content); // разрешаем вложенные шорткоды
    $r = '<div class="message message-' .
        $type . '"><p>' . $content . '</p></div>';
    return $r;
}
add_shortcode('msg', 'sp_msg_shortcode');
?>

```

Обратите внимание, что содержимое, которое вы хотите отобразить, возвращается из функции обратного вызова, а не отражается в буфере вывода. Это связано с тем, что фильтр шорткода обычно запускается до того, как какой-либо контент будет помещен на экран. При наличии каких-либо эхо-вызовов внутри этой функции, вывод был бы показан сверху страницы, а не там, где вы хотите.

Атрибуты шорткода

В предыдущем примере продемонстрирован еще один важный момент — задание атрибутов по умолчанию. Функция `shortcode_atts()` принимает три параметра: `$pairs`, `$atts` и `$shortcode`.

- ◆ `$pairs` — массив атрибутов по умолчанию, где каждый ключ — это имя атрибута, а каждое значение — значение атрибута.
- ◆ `$atts` — это аналогичный массив атрибутов, обычно получаемый прямо из параметра `$atts`, переданного в функцию обратного вызова шорткода. Функция `shortcode_atts()` объединяет атрибуты по умолчанию и переданные атрибуты в один массив.
- ◆ `$shortcode` — необязательный параметр. Если установлено совпадение с именем шорткода, запускается фильтр `shortcode_atts_{shortcode}`, который может использоваться другими плагинами (или чем-то еще) для переопределения атрибутов по умолчанию.

Результаты функции `shortcode_atts()` затем передаются в РНР-функцию `extract()`, которая создает переменную в локальной области видимости для каждого ключа в массиве атрибутов.

Таким образом, переменная `$type` в нашем примере доступна для остальной части функции и содержит либо значение по умолчанию для сообщения, либо любое другое значение, установленное в самом шорткоде.

Вложенные шорткоды

Наконец, мы пропускаем внутреннее содержимое `$content` через функцию `do_shortcode()`, чтобы включить вложенные шорткоды. Если у вас был шорткод `[help_link]`, который генерировал ссылку на вашу документацию, в зависимости от того, на каком разделе сайта вы работали, или под каким типом пользователя вы вошли в систему, вы можете использовать этот шорткод в шорткоде `[msg]`:

```
[msg type="error"]
```

Произошла ошибка. Используйте следующую ссылку для справки: `[help_link]`.

```
[/msg]
```

Пока функция обратного вызова для шорткода `[msg]` передает свои результаты через функцию `do_shortcode()`, внутренний шорткод `[help_link]` будет отфильтрован как предполагалось.



Несмотря на то, что вложенные шорткоды разных типов будут работать, вложение одного и того же шорткода внутрь самого себя вызовет ошибку. Парсер регулярных выражений (regex), который извлекает шорткоды из контента, спроектирован для увеличения скорости выполнения. Парсер должен проходить по содержимому только один раз. Для обработки вложенных шорткодов одного и того же типа потребовалось бы несколько проходов через контент, что замедлило бы алгоритм. Возможные решения этой проблемы: 1) избежать вложения одного и того же шорткода в самого себя; 2) задать шорткодам, которые ссылаются на одну и ту же функцию обратного вызова, разные имена; 3) написать собственный синтаксический анализатор регулярных выражений для вашего шорткода и проанализировать шорткоды самостоятельно.

Функция `do_shortcode()` позволяет также применять шорткоды к настраиваемым полям, содержимому, извлеченному из пользовательских таблиц, или другому содержимому, которое еще не проходило через фильтр `the_content`. В большинстве случаев вне функций самого обратного вызова шорткода, более целесообразно использовать функцию `apply_filters("the_content", $content)`, которая будет применять все фильтры при срабатывании хука `the_content`, включая фильтр шорткодов, как показано в следующем примере.

Пример

```
<?php
global $post;
$sidebar_content = $post->sidebar_content;
?>
```

```
<div class="post">
    <?php the_content(); ?>
</div>
<div class="sidebar">
    <?php
        //echo do_shortcode($sidebar_content);
        echo apply_filters('the_content', $sidebar_content);
    ?>
</div>
```

Удаление шорткодов

Как и в случае с событиями и фильтрами, вы можете удалить зарегистрированные шорткоды, чтобы они не применялись к определенному сообщению или к содержимому, которое вы передаете непосредственно в код `do_shortcode()` или через фильтр `the_content`. Функция `remove_shortcode()` принимает имя шорткода в качестве единственного параметра и отменяет регистрацию указанного шорткода. Функция `remove_all_shortcodes()` отменяет регистрацию всех шорткодов.



Убедитесь, что вызов функции `remove_shortcode()` выполняется достаточно поздно при работе WordPress, чтобы шорткод, который вы хотите удалить, уже был добавлен. Например, если плагин добавляет шорткод во время действия хука `init` с приоритетом 10, то вам нужно поместить вызов `remove_shortcode()` во время действия `init` с приоритетом 11 или выше или через другой хук, который срабатывает после `init`.

Массив зарегистрированных шорткодов хранится в глобальной переменной `$shortcode_tags`. Иногда полезно сделать копии этой переменной или отредактировать ее напрямую. Например, если вы хотите исключить конкретные шорткоды из определенного фрагмента содержимого, то можно сделать резервную копию всех шорткодов, удалить ненужные шорткоды, применить их, а затем восстановить исходный список шорткодов, что демонстрирует следующий пример.

Пример

```
// Делаем копию оригинального списка шорткодов
global $shortcode_tags;
$original_shortcode_tags = $shortcode_tags;

// удаляем шорткод [msg]
unset($shortcode_tags['msg']);

// выполняем шорткоды и вывод
$content = do_shortcode($content);
echo $content;

// восстанавливаем оригинальные шорткоды
$shortcode_tags = $original_shortcode_tags;
```

Другие полезные функции, связанные с шорткодами

Укажем еще несколько функций, полезных при работе с шорткодами:

- ◆ `shortcode_exists($tag)` — проверяет, был ли зарегистрирован шорткод `$tag`;
- ◆ `has_shortcode($content, $tag)` — проверяет, присутствует ли шорткод `$tag` в переменной `$content`;
- ◆ `shortcode_parse_atts($text)` — извлекает атрибуты из шорткода. Это делается за вас при парсинге любого шорткода, но эта функция может вызываться напрямую, если вы хотите извлечь атрибуты из другого текста, такого как теги HTML или другие шаблоны;
- ◆ `strip_shortcodes($text)` — удаляет все шорткоды из переменной `$text` и заменяет их пустым текстом вместо выполнения функции обратного вызова.

Более подробную информацию об API шорткодов можно найти в Кодексе WordPress (bwwawp.com/shortcode-api).

API виджетов

Виджеты позволяют отображать фрагменты кода и содержимое в различных областях на вашем сайте WordPress. Наиболее типичные случаи — это добавление виджетов на боковую панель или область нижнего колонтитула. Вы всегда можете жестко закодировать эти разделы на веб-сайте, но наличие виджетов позволяет вашим разработчикам перетаскивать их из одной области в другую или изменять свои настройки через страницу виджетов на панели администратора. WordPress поставляется со множеством встроенных виджетов, включая базовый текстовый виджет, показанный на рис. 7.1.

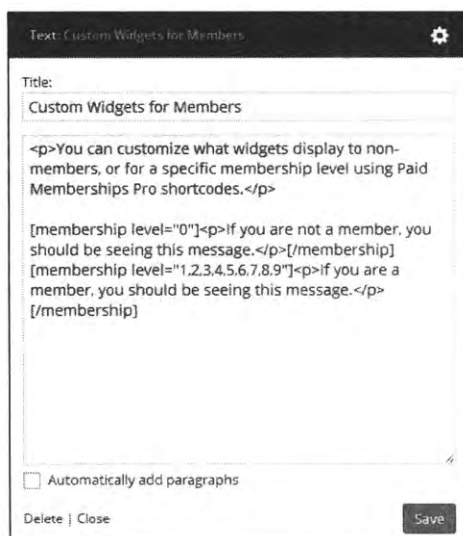


Рис. 7.1. Настройки текстового виджета

Множество плагинов также включает в себя виджеты для отображения различного контента. Здесь мы не будем использовать и стилизовать виджеты, так как это хорошо описано на странице Кодекса WordPress, посвященной виджетам (bwwwp.com/widgets-codex), но рассмотрим, как добавлять виджеты и области виджетов в ваши плагины и темы.



Пользовательский интерфейс страницы виджетов в админ-панели претерпел изменения в WordPress версии 3.8; однако функции и вызовы API для добавления новых виджетов программными средствами не должны сильно изменяться.

Прежде чем добавить свой собственный виджет

Прежде чем приступить к разработке нового виджета, стоит потратить некоторое время, чтобы посмотреть, не подойдет ли вам существующий виджет. Если вы проявите творческий подход, то сможете избежать создания нового виджета.

Найдите в репозитории плагины, в которых уже есть нужный вам виджет. Если это так, дважды проверьте код и посмотрите, будет ли он работать.

Текстовые виджеты могут использоваться для добавления произвольного текста в пространство виджетов. Таким образом вы также можете встроить код JavaScript или добавить шорткод в текстовую область и с его помощью реализовать нужную вам функциональность (возможно, вы создали шорткод уже для другой цели) вместо создания нового виджета.

Если ваш виджет отображает список ссылок, возможно, имеет смысл создать меню из этих ссылок и использовать виджет "Custom Menu", встроенный в WordPress. Другие виджеты, отображающие последние публикации из категории, часто работают с CPT и таксономиями либо сразу "из коробки", либо требуют совсем немного изменений.

Если вам нужен совершенно новый виджет, в следующем разделе описываются необходимые шаги.

Добавление виджетов

Чтобы добавить еще один виджет в WordPress, необходимо создать новый класс PHP для виджета, который расширяет класс `WP_Widget` WordPress. Рекомендуем вам ознакомиться с кодом класса `WP_Widget`, находящимся в файле в `wp-includes/widgets.php`. Комментарии в коде объясняют, как класс работает и какие методы вы должны переопределить, чтобы создать свой собственный класс виджетов. Существуют четыре основных метода, которые вы должны переопределить для класса виджета, что ясно показано в следующем примере кода со страницы Кодекса WordPress, посвященной API виджетов (oreil.ly/ojY06).

Пример

/*

Взято со страницы Кодекса API виджетов по адресу:
http://codex.wordpress.org/Widgets_API

```

*/
class My_Widget extends WP_Widget {

    public function __construct() {
        // настоящие процессы виджета
    }

    public function widget($args, $instance) {
        // выводит содержимое виджета
    }

    public function form($instance) {
        // выводит форму настроек на панели администратора
    }

    public function update($new_instance, $old_instance) {
        // обрабатывает параметры виджета для сохранения
    }
}

add_action('widgets_init', function(){
    register_widget('My_Widget');
});

```

Вызов функции `add_action()` передает анонимную функцию в качестве второго параметра, который поддерживается только в PHP версии 5.3 и выше. Технически, WordPress требует версии PHP 5.2.4 и выше. Альтернатива — использование `create_function()` в PHP, которая медленнее и потенциально менее безопасна, чем анонимная функция. Однако если вы планируете выпустить свой код для широкой аудитории, то можете воспользоваться альтернативным методом, показанным в следующем фрагменте кода:

```

/*
    Взято со страницы Кодекса API виджетов по адресу:
    http://codex.wordpress.org/Widgets_API
*/
add_action('widgets_init',
    create_function('', 'return register_widget("My_Widget");')
);

```

Собрав все воедино, в листинге 7.1 представлен новый виджет для сайта SchoolPress. Этот виджет будет отображать либо глобально определенную заметку, установленную в настройках виджета, либо заметку, относящуюся к текущей группе BuddyPress, установленной администраторами группы.

Листинг 7.1. Виджет заметки SchoolPress

```

<?php
/*
    Виджет для отображения текущей заметки класса.

```

Учителя (администраторы группы) могут изменять заметки для каждой группы. Показывает глобальную заметку, установленную в настройках виджета, если она не пуста.

```
*/
class SchoolPress_Note_Widget extends WP_Widget
{
    public function __construct() {
        parent::__construct(
            'schoolpress_note',
            'SchoolPress Note',
            array('description' => 'Note to Show on Group Pages');
        }

    public function widget($args, $instance) {
        global $current_user;

        // сохранить внесенные изменения?
        if (!empty($_POST['schoolpress_note_text'])
            && !empty($_POST['class_id'])) {
            // убедитесь, что это администратор
            if(groups_is_user_admin($current_user->ID,intval($_POST['class_id']))) {
                // следует экранировать текст и возможно использовать nonce
                update_option(
                    'schoolpress_note_' . intval($_POST['class_id']),
                    $_POST['schoolpress_note_text']
                );
            }
        }

        // ищем глобальную переменную заметки
        $note = $instance['note'];

        // получить идентификатор класса для этой группы
        $class_id = bp_get_current_group_id();

        // ищем заметку для класса
        if (empty($note) && !empty($class_id)) {
            $note = get_option("schoolpress_note_" . $class_id);
        }

        // отобразить заметку
        if (!empty($note)) {
            ?>
            <div id="schoolpress_note">
                <?php echo wpautop($note);?>
            </div>
            <?php
```

```

// показать кнопку редактирования для администраторов группы
if (groups_is_user_admin($current_user->ID, $class_id)) {
    ?>
    <a id="schoolpress_note_edit_trigger">Edit</a>
    <div id="schoolpress_note_edit" style="display: none;">
    <form action="" method="post">
    <input type="hidden"
        name="class_id"
        value="<?php echo intval($class_id);?>" />
    <textarea name="schoolpress_note_text" cols="30" rows="5">
    <?php echo esc_textarea(get_option('schoolpress_note_'.$class_id))
    ;?>
    </textarea>
    <input type="submit" value="Save" />
    <a id="schoolpress_note_edit_cancel" href="javascript:void(0);">
        Cancel
    </a>
    </form>
    </div>
    <script>
    jQuery(document).ready(function() {
        jQuery('#schoolpress_note_edit_trigger').click(function(){
            jQuery('#schoolpress_note').hide();
            jQuery('#schoolpress_note_edit').show();
        });
        jQuery('#schoolpress_note_edit_cancel').click(function(){
            jQuery('#schoolpress_note').show();
            jQuery('#schoolpress_note_edit').hide();
        });
    });
    </script>
    <?php
    }
}

public function form($instance) {
    if (isset($instance['note']))
        $note = $instance['note'];
    else
        $note = "";
    ?>
    <p>
        <label for="<?php echo $this->get_field_id('note'); ?>"
            <?php _e('Note:'); ?>
        </label>
        <textarea id="<?php echo $this->get_field_id('note'); ?>"

```

```

        name="<?php echo $this->get_field_name('note'); ?>"
        <?php echo esc_textarea($note);?>
    </textarea>
</p>
<?php
}

public function update($new_instance, $old_instance) {
    $instance = array();
    $instance['note'] = $new_instance['note'];

    return $instance;
}
}
add_action('widgets_init', function() {
    register_widget('SchoolPress_Note_Widget');
});
?>

```

Определение области виджета

Чтобы добавить области виджетов или боковые панели к вашей теме, вам нужно выполнить две операции. Сначала зарегистрировать область виджетов с помощью WordPress. Затем добавить код к своей теме в том месте, где вы хотите, чтобы область виджета появилась.

Зарегистрировать область виджета довольно просто, вызвав функцию `register_sidebar()`, которая принимает массив аргументов в качестве единственного параметра. Доступны следующие аргументы, взятые со страницы Кодекса WordPress для функции `register_sidebar()` (bwaawwp.com/reg-sidebar):

- ◆ `name` — имя боковой панели (по умолчанию `Sidebar#`, где `#` — идентификатор боковой панели);
- ◆ `id` — идентификатор боковой панели; все символы должны быть в нижнем регистре без пробелов (по умолчанию задан числовой идентификатор с автоинкрементом);
- ◆ `description` — текстовое описание того, что/где находится боковая панель; отображается на экране управления виджетами с версии 2.9 (по умолчанию `empty`);
- ◆ `class` — имя класса CSS для назначения виджетам в HTML (по умолчанию `empty`);
- ◆ `before_widget` — HTML-код, который нужно разместить перед каждым виджетом (по умолчанию `<li id="%1$s" class="widget %2$s">`); использует `sprintf` для подстановки переменных;
- ◆ `after_widget` — HTML-код, который нужно разместить после каждого виджета (по умолчанию `\n`);

- ◆ `before_title` — HTML-код для размещения перед каждым заголовком (по умолчанию `<h2 class="widgettitle">`);
- ◆ `after_title` — HTML-код для размещения после каждого заголовка (по умолчанию `</h2>\n`).

Чтобы зарегистрировать пустую боковую панель для страниц заданий нашей темы SchoolPress, мы добавили бы в файл `functions.php` или `include/sidebars.php` нашей темы код следующего примера.

Пример

```
register_sidebar(array(
    'name' => 'Assignment Pages Sidebar',
    'id' => 'schoolpress_assignment_pages',
    'description' => 'Sidebar used on assignment pages.',
    'before_widget' => '',
    'after_widget' => '',
    'before_title' => '',
    'after_title' => ''
));
```

Значения для `before/after_widget` и `before/after_title` будут установлены на основе того, как наша тема стилизует виджеты и заголовки. Некоторые ожидают элементы ``; другие — элементы `<div>`. Но если все стили обрабатываются кодом нашего виджета, то мы можем просто установить все в пустые строки. Далее нам нужно встроить область виджетов в нашу тему. Мы делаем это с помощью функции `dynamic_sidebar()`, которая принимает идентификатор зарегистрированной боковой панели в качестве единственного параметра:

```
if(!dynamic_sidebar('schoolpress_student_status'))
{
    // резервный код в случае, если боковая панель my_widget_area не найдена
}
```

Код загрузит боковую панель `schoolpress_student_status`, если та найдена. В противном случае функция `dynamic_sidebar()` вернет `false` и вместо появления панели будет выполнен код внутри фигурных скобок. Это позволит отобразить содержимое по умолчанию в области боковой панели, если в области боковой панели нет виджетов или ее нет вообще.

Исторически темы WordPress разрабатывались с областью боковой панели, и темы жестко кодировали в них определенные функции. Виджеты были впервые представлены в первую очередь для замены этих статических боковых панелей динамическими боковыми панелями, которыми можно было управлять через страницу виджетов панели инструментов. Вот почему термин *боковая панель* употребляется для определения областей виджетов, даже если виджеты используются не только на боковых панелях.

Если вам необходимо знать, зарегистрирована ли боковая панель и используется ли она (имеет ли виджеты) без фактического встраивания виджетов, то вы можете прибегнуть к помощи функции `is_active_sidebar()`. Просто передайте идентификатор боковой панели, и функция вернет `true`, если на боковой панели находится зарегистрированный виджет или `false`, если это не так. Тема "Twenty Thirteen" использует эту функцию, чтобы проверить наличие виджетов на боковой панели перед отображением HTML-кода для боковой панели, что иллюстрирует следующий пример.

Пример

```
<?php
// из файла twenty-thirteen/sidebar.php
if (is_active_sidebar('sidebar-2')) :>
    <div id="tertiary" class="sidebar-container" role="complementary">
        <div class="sidebar-inner">
            <div class="widget-area">
                <?php dynamic_sidebar('sidebar-2'); ?>
            </div><!-- .widget-area -->
        </div><!-- .sidebar-inner -->
    </div><!-- #tertiary -->
<?php endif; ?>
```

Встраивание виджета вне динамической боковой панели

Обычный процесс добавления виджетов на ваши страницы описан в предыдущем разделе, где вы определяете динамическую боковую панель, а затем добавляете свой виджет на боковую панель через страницу виджетов на панели администратора.

В качестве альтернативы, если вы точно знаете, какой виджет хотите включить в какое-либо место, и не хотите, чтобы размещение виджета оставалось на усмотрение администраторов, управляющих настройками виджетов на панели инструментов, вы можете встроить виджет с помощью функции `the_widget($widget, $instance, $args)` с такими аргументами:

- ◆ `$widget` — имя класса PHP для вашего виджета;
- ◆ `$instance` — массив, содержащий настройки для вашего виджета;
- ◆ `$args` — массив, содержащий аргументы, которые обычно передаются в `register_sidebar()`.

Помимо жесткого кодирования размещения виджета, применение функции `the_widget()` также позволяет программно устанавливать настройки виджета. В следующем примере мы встраиваем виджет StudentPress Note непосредственно на страницу темы. Мы устанавливаем массив экземпляров, чтобы включить пустую строку для значения `$note`, гарантируя, что примечание группы отображается, если доступно.

```
// Показать виджет заметки, переопределяя глобальную заметку the_widget
the_widget('SchoolPress_Note_Widget',      // имя класса
    array('note'=> ''),                      // переменные экземпляра
    array(                                    // переменные виджета
        'before_widget' => '',
        'after_widget' => '',
        'before_title' => '',
        'after_title' => ''
    )
);
```

API виджетов панели инструментов

Виджеты панели инструментов — это поля, которые отображаются на домашней странице панели администратора WordPress (рис. 7.2).

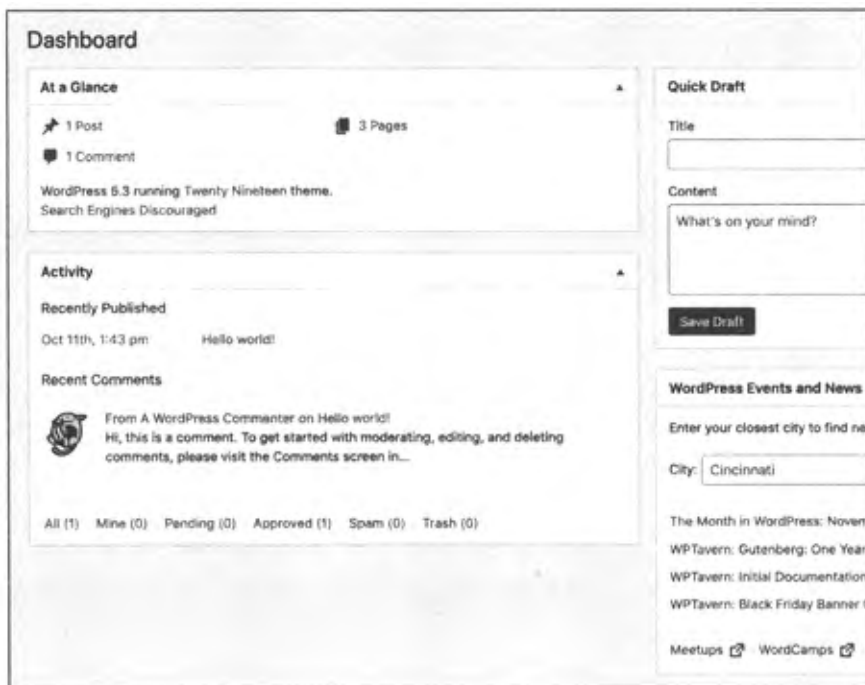


Рис. 7.2. Виджеты на панели администрирования

По умолчанию WordPress включает в себя несколько разных виджетов панели инструментов. Добавляя и удаляя виджеты с этой панели с помощью API виджетов, вы можете сделать приложение WordPress более полезным, разместив информацию

и инструменты, наиболее необходимые для вашего приложения, непосредственно на домашней странице панели администрирования. И в идеале это должно быть сделано для всех приложений WordPress с пользователями, которые будут обращаться к административной панели WordPress.

Удаление виджетов с панели инструментов

Виджеты панели администрирования — это на самом деле просто метаблоки, назначенные странице панели инструментов администратора. На странице API виджетов панели инструментов Кодекса WordPress (oreil.ly/OcKZJ) перечислены списки виджетов по умолчанию, отображаемых на панели инструментов WordPress, что демонстрирует следующий пример.

Пример

```
// со страницы кодекса API виджетов панели инструментов
// Основной столбец:
$wp_meta_boxes['dashboard']['normal']['high']['dashboard_browser_nag']
$wp_meta_boxes['dashboard']['normal']['core']['dashboard_right_now']
$wp_meta_boxes['dashboard']['normal']['core']['dashboard_recent_comments']
]
$wp_meta_boxes['dashboard']['normal']['core']['dashboard_incoming_links']
$wp_meta_boxes['dashboard']['normal']['core']['dashboard_plugins']

// Боковой столбец:
$wp_meta_boxes['dashboard']['side']['core']['dashboard_quick_press']
$wp_meta_boxes['dashboard']['side']['core']['dashboard_recent_drafts']
$wp_meta_boxes['dashboard']['side']['core']['dashboard_primary']
$wp_meta_boxes['dashboard']['side']['core']['dashboard_secondary']
```

Чтобы удалить виджеты с панели инструментов, вы можете вызвать функцию `remove_meta_box($id, $page, $context)`:

- ♦ `$id` — идентификатор, определенный при добавлении метаблока. Он установлен как атрибут `id` элемента `<div>`, созданного для метаблока;
- ♦ `$page` — имя страницы администратора, к которой был добавлен метаблок. Используйте `dashboard`, чтобы удалить метаблоки панели инструментов;
- ♦ `$context` — `normal` (обычный), `advanced` (расширенный) или `side` (боковой), в зависимости от того, где был добавлен метаблок.

Чтобы удалить все виджеты по умолчанию, вы можете подключиться к `wp_dashboard_setup` и вызвать функцию `remove_meta_box()` для каждого виджета, который вы хотите удалить, как показано в следующем примере.

Пример

```
// Удалить все стандартные виджеты WordPress с панели инструментов
function sp_remove_dashboard_widgets()
{
    remove_meta_box('dashboard_browser_nag', 'dashboard', 'normal');
    remove_meta_box('dashboard_right_now', 'dashboard', 'normal');
    remove_meta_box('dashboard_recent_comments', 'dashboard', 'normal');
    remove_meta_box('dashboard_incoming_links', 'dashboard', 'normal');
    remove_meta_box('dashboard_plugins', 'dashboard', 'normal');

    remove_meta_box('dashboard_quick_press', 'dashboard', 'side');
    remove_meta_box('dashboard_recent_drafts', 'dashboard', 'side');
    remove_meta_box('dashboard_primary', 'dashboard', 'side');
    remove_meta_box('dashboard_secondary', 'dashboard', 'side');
}
add_action('wp_dashboard_setup', 'sp_remove_dashboard_widgets');
```

На многосайтовой сетевой панели создается другой набор виджетов, и вы должны использовать другой хук для удаления виджетов сетевой панели управления. Код в следующем примере перехватывает `wp_network_dashboard_setup` и удаляет метаблоки, добавленные на страницу `$dashboard-network`.

Пример

```
// Удалить сетевые виджеты
function sp_remove_network_dashboard_widgets()
{
    remove_meta_box('network_dashboard_right_now', 'dashboard-network',
'normal');
    remove_meta_box('dashboard_plugins', 'dashboard-network', 'normal');
    remove_meta_box('dashboard_primary', 'dashboard-network', 'side');
    remove_meta_box('dashboard_secondary', 'dashboard-network', 'side');
}
add_action('wp_network_dashboard_setup',
'sp_remove_network_dashboard_widgets');
```

Подобный код подойдет для удаления метаблоков по умолчанию с других страниц панели мониторинга, таких как страница редактирования и страницы публикации. Значение `$page`, используемое при удалении метаполей, есть `page` и `post` соответственно.

Добавление собственного виджета на панель инструментов

Функция `wp_add_dashboard_widget()` является оболочкой для `add_meta_box()`, которая добавляет виджет на страницу панели администратора. Функция `wp_add_dashboard_widget()` принимает четыре параметра:

- ◆ `$widget_id` — идентификатор для ваших виджетов, который добавляется в качестве имени класса CSS в оболочку для виджета, а также используется в качестве ключа массива для массива виджетов на панели мониторинга;
- ◆ `$widget_name` — имя виджета, которое отображается в заголовке виджета;
- ◆ `$callback` — функция обратного вызова, которая отображает виджет;
- ◆ `$control_callback` — необязательный параметр. Значение по умолчанию `null`. Задает функцию обратного вызова для обработки отображения и страницы конфигурации виджета.

В листинге 7.2 добавляется виджет сводной панели для отображения статуса текущих заданий (рис. 7.3). Код включает в себя вызов функции `wp_add_dashboard_widget()` для регистрации виджета на панели администрирования, а также функцию обратного вызова для отображения этого фактического виджета и другую функцию обратного вызова для обработки предоставленной конфигурации виджета (рис. 7.4).

Листинг 7.2. Виджет панели заданий

```
<?php
/*
 * Добавляем виджеты на панель администрирования
 */
function sp_add_dashboard_widgets() {
    wp_add_dashboard_widget(
        'schoolpress_assignments',
        'Assignments',
        'sp_assignments_dashboard_widget',
        'sp_assignments_dashboard_widget_configuration'
    );
}
add_action('wp_dashboard_setup', 'sp_add_dashboard_widgets');

/*
 * Виджет панели заданий
 */
// виджет
function sp_assignments_dashboard_widget() {
    $options = get_option("assignments_dashboard_widget_options", array());

    if (!empty($options['course_id'])) {
        $group = groups_get_group(array(
```

```

        'group_id'=>$options['course_id']
    ));
}

if (!empty($group)) {
    echo "Showing assignments for class " .
        $group->name . "<br />...";
    /*
        получить задания для этой группы и перечислить их статус
    */
}
else {
    echo "Showing all assignments.<br />...";
    /*
        получить все задания и перечислить их статус
    */
}
}

// настройки
function sp_assignments_dashboard_widget_configuration() {
    // получить старые настройки или пустой массив,
    // если все значения заданы по умолчанию
    $options = get_option("assignments_dashboard_widget_options", array());

    // Сохранить настройки ?
    if (isset($_POST['assignments_dashboard_options_save'])) {
        // получаем course_id
        $options['course_id'] = intval(
            $_POST['assignments_dashboard_course_id']
        );

        // сохранить
        update_option("assignments_dashboard_widget_options", $options);
    }

    // показать форму параметров
    $groups = groups_get_groups(array('orderby'=>'name', 'order'=>'ASC'));
    ?>
    <p>Choose a class/group to show assignments from.</p>
    <div class="feature_post_class_wrap">
        <label>Class</label>
        <select name="assignments_dashboard_course_id">
            <option value="" <?php selected($options['course_id'], "");?>
                All Classes
            </option>
            <?php
                $groups = groups_get_groups(array('orderby'=>'name',
                    'order'=>'ASC'));

```

```

        if (!empty($groups) && !empty($groups['groups'])) {
            foreach ($groups['groups'] as $group) {
                ?>
                <option value="<?php echo intval($group->id);?>"
                <?php selected($options['course_id'], $group->id);?>"
                <?php echo $group->name;?>
                </option>
                <?php
            }
        }
        ?>
    </select>
</div>
<input type="hidden" name="assignments_dashboard_options_save" value="1"
/>
<?php
}
?>

```

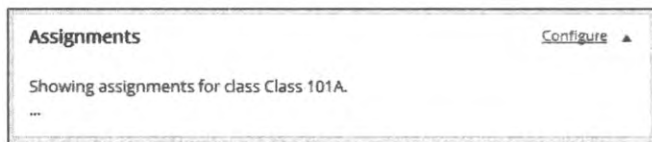


Рис. 7.3. Наш виджет заданий

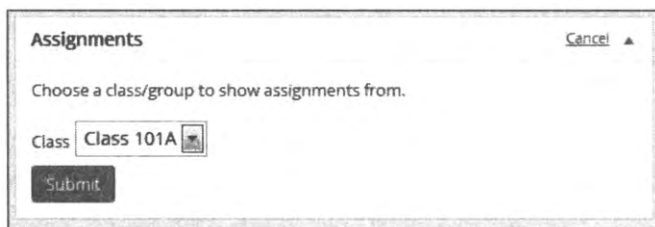


Рис. 7.4. Вид конфигурации нашего виджета заданий

Обратите внимание, что мы использовали хук `wp_dashboard_setup` для вызова функции, которая добавляет наш виджет. Если бы мы хотели, чтобы наш виджет отображался на информационной панели сетевого сайта, то нам потребовался бы хук `wp_network_dashboard_setup`.

Функция `sp_assignments_dashboard_widget()` рисует фактический виджет, отображаемый на странице панели инструментов. Здесь мы добавили бы наш код, чтобы пройти по циклу заданий и показать статистику того, какой процент заданий был сдан.

Функция `sp_assignments_dashboard_widget_configuration()` выводит форму конфигурации, а также содержит код для обработки отправки формы и обновления опции, в которой мы сохраняем настройки.

API настроек

WordPress предлагает API, с помощью которого вы можете создавать опции и формы настроек для ваших плагинов на панели администратора. Этот API настроек очень тщательно документирован в Кодексе WordPress (bawawp.com/settings-api). У Тома Макфарлина также есть отличное учебное пособие по настройке API WordPress на Envato Tutsplus (bawawp.com/guide-wp). В этих источниках изложены детали добавления страниц меню и настроек в них для использования в ваших плагинах и темах. Далее приведены некоторые советы специально для разработчиков приложений.

Вам действительно нужна страница настроек?

Прежде чем тратить время на создание страницы настроек и усложнять ваше приложение, рассмотрите возможность использования глобальной переменной для хранения массива параметров, необходимых вашему плагину или приложению:

```
global $schoolpress_settings;  
$schoolpress_settings = array(  
    'info_email' => 'info@schoolpress.me',  
    'info_email_name' => 'SchoolPress'  
);
```

Для приложений, которые не будут управляться обычными пользователями и/или не будут распространяться, глобальной переменной вполне достаточно для хранения ваших настроек. Просто сохраните глобальную переменную как в предыдущем фрагменте кода в начале файла плагина или в файле `includes/settings.php`. Зачем создавать пользовательский интерфейс, если вы не собираетесь его применять?

Даже если ваш плагин или тема будут в конечном итоге использовать другие, мы все равно хотели бы начать с такой глобальной переменной. Настройки, которые, по вашему мнению, вам нужны в начале, могут не соответствовать тем, которые вам понадобятся в конце вашего проекта. Настройки могут добавляться или удаляться во время разработки. Для настроек, которым, по вашему мнению, требуется раскрывающийся список, может в итоге понадобиться просто поле свободного текста. API настроек позволяет легко добавлять настройки и обновлять их позже, но все же гораздо проще изменить один элемент в глобальном массиве, чем добавить или изменить несколько вызовов функций и определений.

Если к вам применимо большинство следующих утверждений, рассмотрите возможность использования глобальной переменной для ваших настроек вместо создания пользовательского интерфейса настроек:

- ◆ Этот плагин не будет распространяться за пределами моей команды.
- ◆ Единственные люди, которые меняют эти настройки, — разработчики.
- ◆ Эти настройки должны быть одинаковыми в разных средах.
- ◆ Эти настройки могут измениться до выпуска продукта.

Не могли бы вы использовать вместо этого хук или фильтр?

Еще одна альтернатива добавлению настроек в ваш плагин через API настроек — использовать хук или фильтр. Если настройка, которую вы задумали, будет нужна только ограниченному числу ваших пользователей, рассмотрите возможность добавления хука или фильтра для облегчения разработки.

Например, кто-то, использующий наш плагин WP-Дос, может запросить возможность ограничить создание файлов *.doc только администраторами или определенным подмножеством ролей WordPress. Мы могли бы добавить страницу настроек со списком ролей с флажками, чтобы включить или отключить возможность загрузки файлов *.doc для этой роли. Возможно, понадобится только один флажок, чтобы разрешить загрузку для всех ролей или только для администраторов. Может быть, это будет свободное текстовое поле для ввода имени возможности, которую нужно проверить перед разрешением загрузки.

Фильтр может оказаться лучшим решением проблемы. Мы можем добавить проверку возможностей до того, как файл *.doc загружен, и задействовать фильтр, чтобы позволить разработчикам переопределить массив возможностей по умолчанию, которое проверяются. Этот код должен быть добавлен в функцию `wpdoc_template_redirect()` плагина WP-Дос до отображения страницы *.doc: как показано в следующем примере.

Пример

```
// По умолчанию не требуется никаких возможностей,
// но разработчики могут добавлять проверки
$caps = apply_filters('wpdoc_caps', array());

if(!empty($caps))
{
    // Виновен, если не докажет иное
    $hascap = false;

    // необходимо войти в систему, чтобы иметь какие-либо возможности
    if(is_user_logged_in())
    {
        // убедитесь, что у текущего пользователя есть одна из возможностей
        foreach($caps as $cap)
        {
            if(current_user_can($cap))
            {
                $hascap = true;
                break; // прекратить проверку
            }
        }
    }
}
```

```

if (!$hascap)
{
    // не показывать им файл
    header('HTTP/1.1 503 Service Unavailable', true, 503);
    echo "HTTP/1.1 503 Service Unavailable";
    exit;
}
}

```

Затем вы можете переопределить массив `wpdoc_caps`, добавив такие действия:

```

// требуется любая учетная запись пользователя
add_filter('wpdoc_caps', function($caps) { return array('read'); });
// требуется учетная запись администратора
add_filter('wpdoc_caps', function($caps) { return array('manage_options'); });
// только авторы или пользователи с пользовательской возможностью (doc)
add_filter('wpdoc_caps', function($caps) { return array('edit_post', 'doc'); });

```



В предыдущем примере присутствуют анонимные функции, также известные как **замыкания**, поэтому вызов `add_filter()` можно записать в одну строку без использования отдельной функции обратного вызова. Этот синтаксис требует PHP версии 5.3 или выше.

Напомним, что чем больше число следующих утверждений, которые являются истинными, тем больше имеет смысл заменить пользовательский интерфейс настроек хуком или фильтром:

- ◆ Немногие пользователи захотят изменить эту настройку.
- ◆ Люди, меняющие этот параметр, скорее всего, будут разработчиками.
- ◆ Люди, изменяющие этот параметр, могут иметь собственные потребности.
- ◆ Для этого параметра потребуется большое число отдельных настроек или более гибкий UI.

Учет стандартов при добавлении настроек

Если вам необходимо добавить настройки для своего плагина или темы, обратитесь к учебным пособиям, перечисленным ранее в этой главе, чтобы убедиться, что вы правильно используете API настроек.

Применение API настроек требует некоторой предварительной работы, но позволяет легче добавлять и редактировать настройки в дальнейшем. И если вы следуете стилю WordPress, другие разработчики поймут, как работает ваша программа, и смогут подключиться к ней. Если разработчик захочет создать надстройку для вашего плагина, он сможет подключиться к существующим меню и разделам настроек, чтобы добавить новые настройки для своих плагинов.

API настроек также гарантирует, что ваши настройки будут выглядеть аналогично другим настройкам на пользовательской панели WordPress. Мы не хотим, чтобы

разработчикам пришлось изучать новый пользовательский интерфейс для работы с вашим плагином.

Игнорирование стандартов при добавлении настроек

Хотя обычно лучше использовать API настроек и стандарты WordPress при добавлении настроек для своего плагина, иногда имеет смысл игнорировать эти стандарты.

Наиболее распространенная ситуация — наличие у вас многочисленных настроек, которые заслуживают особого пользовательского интерфейса. Если у вас есть только одна или две настройки, пользователи не будут тратить много времени на экранах настроек. Они просто захотят изменить эти две настройки как можно быстрее.

Однако, если вашему плагину требуются десятки настроек, может быть на нескольких вкладках или экранах, возможно, связанных друг с другом, то имеет смысл рассматривать настройки вашего приложения как самостоятельное приложение. Вы должны уделить некоторое внимание тому, чтобы UI и UX для экрана настроек были максимально оптимизированы.

API настроек WordPress довольно гибок в плане отображения объектов. Вы можете управлять тем, как отображается каждый раздел, и каким образом отображается каждое отдельное поле настроек. Но, в конце концов, он действительно ориентирован на одну или несколько вкладок с разделами и полями на них. Для приложений с большим количеством настроек, которые взаимодействуют друг с другом, целесообразнее другая организация.

Не бойтесь игнорировать стандарты здесь. Добавьте меню на панель инструментов, включите функцию обратного вызова для него, включите набор упорядоченных PHP-файлов для создания формы настроек и ее обработки, и, если возможно, следуйте этим советам:

- ◆ Добавьте разделы и пункты меню в соответствии со стандартами, даже если сами страницы настроек имеют собственный макет.
- ◆ Не забудьте обработать свои входные данные и использовать `nonce`, когда это необходимо.
- ◆ Применяйте хуки и фильтры, когда это возможно, если вы хотите, чтобы другие могли расширить ваши настройки.
- ◆ По возможности выбирайте одинаковые элементы HTML и классы CSS, чтобы общий стиль оставался согласованным с остальной частью WordPress сейчас и в будущих обновлениях.

Из-за сложности программного обеспечения интернет-магазина логично, что плагины электронной коммерции часто имеют непростые экраны настроек. Вот два примера плагинов, где хорошо выполнены пользовательские страницы настроек:

- ◆ Paid Memberships Pro (bwawwp.com/paid-pro) (код опубликован на GitHub: [bwawwp.com/pmp-github](https://github.com/bwawwp/pmp-github)).
- ◆ WooCommerce (bwawwp.com/wcomm-plugin) (код опубликован на GitHub: [bwawwp.com/wc-github](https://github.com/bwawwp/wc-github)).

API перезаписи

Apache поставляется с удобным модулем под названием `mod_rewrite`, который позволяет маршрутизировать входящие URL-запросы на разные URL-адреса или местоположения файлов, используя правила, которые обычно добавляются в файл `*.htaccess` в корневой папке вашего сайта. Другие веб-серверы имеют аналогичные системы перезаписи URL. Вот пример, иллюстрирующий стандартные правила для WordPress:

Пример

```
# BEGIN WordPress
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase /
RewriteRule ^index\.php$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.php [L]
</IfModule>
# END WordPress
```

Если вы хотите больше узнать о модуле Apache `mod_rewrite` и о том, как работают правила WordPress, Дэвид Уолш предлагает отличное построчное объяснение (bwwwp.com/htaccess) файла WordPress `*.htaccess` в своем блоге. Обычно, эти правила перенаправляют весь входящий трафик на *любой* URL-адрес, который не принадлежит ни каталогу, ни файлу, в файл `index.php` вашей инсталляции WordPress.

Затем WordPress анализирует фактический URL-адрес, чтобы определить, какую публикацию, страницу или другую ссылку нужно показать. Например, в большинстве параметров постоянных ссылок URL-адрес `/about/` будет перенаправлять на страницу или сообщение со слагом "about".

По большей части вы можете позволить WordPress делать свое дело и обрабатывать перенаправления постоянных ссылок самостоятельно. Но если вам нужно добавить свои собственные правила для обработки определенных URL-адресов особым образом, то вы можете сделать это через API Rewrite.

Добавление правил перезаписи

Основная функция для добавления правила перезаписи — `add_rewrite_rule($ rule, $ rewrite, $position)`:

- ◆ `$rule` — регулярное выражение для сопоставления с URL-адресом, аналогичное правилу перезаписи Apache;

- ◆ `$rewrite` — URL, на который необходимо перезаписать, если правило соответствует. Совпадающие группы из регулярного выражения правила содержатся в массиве с именем `$match`;
- ◆ `$position` — указывает, следует ли размещать правила над правилами WordPress по умолчанию (вверху) или под ними (внизу).

Если вы хотите передать строку темы в контактную форму через URL-адрес, то можете использовать URL-адрес, например `/contact/special-offer/`, который будет загружать страницу контактов и предварительно указывать тему специального предложения. Вам нужно добавить правило перезаписи, например такое, как в следующем примере.

Пример

```
add_rewrite_rule(
    '/contact/([~/]+)/?',
    'index.php?name=contact&subject=' . $matches[1],
    'top'
);
add_rewrite_rule(
flush_rewrite_rules();
```

Если добавить этот код к правилам перезаписи, то при посещении страницы `/contact/special-offer/` вы будете перенаправлены на страницу `/contact/` и значением глобальной переменной `$wp_query-> query_vars['subject']` станет текст специального предложения или любой другой, который был добавлен после `/contact/`. Ваша контактная форма может использовать это значение для предварительного заполнения значения темы электронного письма.

Сброс правил перезаписи

WordPress кэширует правила перезаписи. Поэтому, когда вы добавляете такое правило, вам нужно сбросить все правила перезаписи, чтобы они вступили в силу заново. Очистка правил перезаписи может занять некоторое время, поэтому важно, чтобы это не происходило при каждой загрузке страницы. Чтобы сохранить правила перезаписи в нужном порядке, в каждом плагине, который их затрагивает, должны выполняться следующие операции:

- ◆ Добавьте правило во время активации плагина и сразу же сбросьте правила перезаписи, вызвав функцию `flush_rewrite_rules()`.
- ◆ Добавьте правило во время хука инициализации в случае, если правила сбрасываются вручную через страницу настроек постоянных ссылок панели администратора или с помощью другого плагина.
- ◆ Добавьте вызов `flush_rewrite_rules()` во время деактивации, чтобы правило удалялось при деактивации.

Следующий пример показывает, как следует добавить наше правило темы в форму обратной связи в соответствии с тремя предыдущими шагами:

Пример

```
// Добавить правило и сбросить все при активации.
function sp_activation()
{
    add_rewrite_rule(
        '/contact/([^\/]+)/?',
        'index.php?name=contact&subject=' . $matches[1],
        'top'
    );
    flush_rewrite_rules();
}
register_activation_hook(__FILE__, 'sp_activation');

/*
    Добавить правило в хук init в случае, если другой плагин решит сбросить правила
    перезаписи, но не делать это самому, так как это накладно
*/
function sp_init()
{
    add_rewrite_rule(
        '/contact/([^\/]+)/?',
        'index.php?name=contact&subject=' . $matches[1],
        'top'
    );
}
add_action('init', 'sp_init');

// Сбросить правила перезаписи при деактивации, чтобы удалить наше правило.
function sp_deactivation()
{
    flush_rewrite_rules();
}
register_deactivation_hook(__FILE__, 'sp_deactivation');
```

Другие функции перезаписи

WordPress предлагает некоторые другие функции для создания специальных видов правил перезаписи:

- ◆ `add_rewrite_tag()` (bwwawp.com/rewrite-api) — еще один способ добавления пользовательских переменных к запросам;
- ◆ `add_feed()` (bwwawp.com/rewrite-add) — добавляет новый вид канала, который будет функционировать как RSS и ATOM каналы;

- ◆ `add_rewrite_endpoint (bwawwp.com/rewrite-end)`. — добавляет переменные запроса к концу URL.

На страницах Кодекса для каждой функции есть хорошее объяснение. Некоторые функции предпочтительнее для конкретных целей по сравнению с другими. В листинге 7.3 показано, как использовать функцию `add_rewrite_endpoint()` для определения, когда к концу URL добавляется `/doc/`, чтобы автоматически скачать файл формата DOC. Здесь учтен тот факт, что любой документ HTML с расширением `doc` будет прочитан Microsoft Word как файл формата DOC.

Функция `add_rewrite_endpoint()` принимает два параметра:

- ◆ `$name*` — имя конечной точки, например, `"doc"`;
- ◆ `$places*` — указывает, к каким страницам добавить правило конечной точки. Использует константы `EP_*`, определенные в файле `wp-includes/rewrite.php`.

Листинг 7.3. Плагин WP DOC

```
<?php
/*
Plugin Name: WP DOC
Plugin URI: http://bwawwp.com/wp-docx/
Description: Добавьте /doc/ в конец страницы или публикации, чтобы скачать версию
*.docx.
Version: .1
Author: Stranger Studios
*/
/*
    Зарегистрировать конечную точку перезаписи
*/
// Добавить конечную точку /doc/ при активации.
function wpdoc_activation()
{
    add_rewrite_endpoint('doc', EP_PERMALINK | EP_PAGES);
    flush_rewrite_rules();
}
register_activation_hook(__FILE__, 'wpdoc_activation');

// И добавить правило в хук init в случае, если другой плагин решит сбросить
// правила перезаписи, но не делать это самому, так как это накладно
function wpdoc_init()
{
    add_rewrite_endpoint('doc', EP_PERMALINK | EP_PAGES);
}
add_action('init', 'wpdoc_init');

// Сбросить правила перезаписи при деактивации, чтобы удалить нашу конечную точку.
function wpdoc_deactivation()
```



```

{
    flush_rewrite_rules();
}
register_deactivation_hook(__FILE__, 'wpdoc_deactivation');

/*
    Определить использование /doc/ и вернуть файл *.doc.
*/
function wpdoc_template_redirect()
{
    global $wp_query;
    if(isset($wp_query->query_vars['doc']))
    {
        global $post;

        // перепроверяем, что это публикация
        if(empty($post->ID))
            return;

        // заголовки для MS Word
        header("Content-type: application/vnd.ms-word");
        header('Content-Disposition: attachment;Filename="'.
            $post->post_name.'.doc');

        //html
        ?>
        <html>
        <body>
        <h1><?php echo $post->post_title; ?></h1>
        <?php
        echo apply_filters('the_content', $post->post_content);
        ?>
        </body>
        </html>
        <?php

        exit;
    }
}
add_action('template_redirect', 'wpdoc_template_redirect');
?>

```

Обратите внимание, что в предыдущем листинге мы соблюдаем три шага, которые описаны в примере с функцией `add_rewrite_rule()`, чтобы определить наше правило при активации и инициализации и сбросить все правила при активации и деактивации.

Мы использовали маски `EP_PERMALINK` | `EP_PAGES` при определении нашей конечной точки, которая добавит ее к отдельным страницам постов и страницам страниц¹. Полный список масок конечных точек выглядит следующим образом:

Примеры масок

```
EP_NONE
EP_PERMALINK
EP_ATTACHMENT
EP_DATE
EP_YEAR
EP_MONTH
EP_DAY
EP_ROOT
EP_COMMENTS
EP_SEARCH
EP_CATEGORIES
EP_TAGS
EP_AUTHORS
EP_PAGES
EP_ALL
```

Дополнительную информацию об API Rewrite можно найти на странице Кодекса API Rewrite (bwwawp.com/rewrite-codex) и на странице Кодекса класса `WP_Rewrite` (bit.ly/class-ref-wp). Они являются хорошими источниками информации. Класс `WP_Rewrite` позволяет сделать гораздо больше, мы до этого просто не дошли.

Функция *WP-Cron*

Задание Cron — это скрипт, который запускается на сервере через определенные интервалы. Функции `WP-Cron` в WordPress расширяют эту функциональность для вашего сайта WordPress. Задания Cron, иногда называемые *событиями*, можно настроить на запуск каждые несколько минут, каждые несколько часов, каждый день или в определенные дни недели или месяца. Некоторые типичные применения заданий Cron включают в себя разбор очередей дайджест-писем, синхронизацию данных со сторонними API-интерфейсами и предварительную обработку вычислений с интенсивной загрузкой ЦП, применяемую в отчетах и сравнительном анализе.

При добавлении задания Cron в ваше приложение есть три основных части:

- ◆ Запланируйте событие Cron. Оно будет запускать определенный хук/действие через определенный интервал времени.

¹ Посты со страницы `post_type`.

- ◆ Подключите функцию к этому действию.
- ◆ Поместите код, который вы действительно хотите запустить, в функцию обратного вызова.

Этот код можно добавить в пользовательский файл плагина для планирования некоторых заданий Cron, как показано в следующем примере².

Пример

```
// Планируем задание Cron на функцию активации плагина sp_activation()
function sp_activation()
{
    // do_action('sp_daily_cron'); будет запускать ежедневно
    wp_schedule_event(time(), 'daily', 'sp_daily_cron');
}

register_activation_hook(__FILE__, 'sp_activation');

// очищаем наши задания cron при вызове функции деактивации плагина
function sp_deactivation()
{
    wp_clear_scheduled_hook('sp_daily_cron');
}

register_deactivation_hook(__FILE__, 'sp_deactivation');

// функция для ежедневного запуска
function sp_daily_cron()
{
    // делаем это ежедневно
}

add_action("sp_daily_cron", "sp_daily_cron");
```

Атрибуты функции `wp_schedule_event(timestamp, $recurrence, $hook, $args)`:

- ◆ `$timestamp` — отметка времени для первого запуска Cron. Обычно вы можете установить его в значение `time()`;
- ◆ `$recurrence` — параметр, определяющий, как часто должно проходить событие. Оно может проходить ежечасно, ежедневно или дважды в день. Хук `cron_schedules` дает возможность добавить другие интервалы;
- ◆ `$hook` — название действия, которое необходимо производить при каждом повторении;
- ◆ `$args` — любые аргументы, которые вы хотели бы передать обработчику, могут быть добавлены в конец вызова `wp_schedule_event()`.

² Если вы переместите этот код в подкаталог вашего плагина, то необходимо обновить вызовы `register_activation_hook()` и `register_deactivation_hook()`, чтобы они указывали на основной файл плагина.

Нам нравится давать нашим событиям Cron имена, основанные на интервале. Таким образом, если мы хотим запускать другую функцию ежедневно, то можем просто добавить строку `add_action('sp_daily_cron', 'new_function_name');` в наш основной код.

Добавление своих интервалов

По умолчанию функция `wp_schedule_event()` принимает только интервалы ежедневно, ежедневно или дважды в день. Чтобы добавить другие интервалы, вам потребуется хук `cron_schedules`, что иллюстрирует следующий пример.

Пример

```
// Добавить месячный интервал для использования в функциях заданий Cron
function sp_cron_schedules($schedules)
{
    $schedules['monthly'] = array(
        'interval' => 60*60*24*30, // на самом деле раз в 30 дней
        'display' => 'Once a Month'
    );
}
add_filter('cron_schedules', 'sp_cron_schedules');
```

В отличие от заданий Cron в Unix, WP-Cron не поддерживает интервалы, основанные на дне недели. Для этого вы можете использовать ежедневное задание Cron и вызывать функцию проверки дня недели, как показано в следующем примере.

Пример

```
// Запускать по понедельникам
function sp_monday_cron()
{
    // получить день недели, 0-6, начиная с воскресенья
    $weekday = date("w");

    // это понедельник?
    if($weekday == "1")
    {
        // выполнять этот код по понедельникам
    }
}
add_action("sp_daily_cron", "sp_monday_cron");
```

Вы можете написать аналогичный код для проверки определенного дня месяца (`date("j")`) или даже конкретных месяцев (`date("m")`).

Планирование единичных событий

В предыдущих примерах показано, как выполнять код с определенным интервалом. У вас также могут быть случаи, когда вы хотите запустить событие один раз в какой-то момент в будущем. Например, вы можете запланировать доставку по электронной почте новых сообщений в блоге через час после их публикации. Это даст авторам час времени, чтобы исправить любые проблемы с сообщением в блоге, прежде чем они будут распространены по всему миру. Для тех случаев, когда мы хотим запланировать запуск события только один раз, подойдет функция `wp_schedule_single_event()`.

Запуск заданий Cron с сервера

Во всех предыдущих примерах мы предполагали, что события, запланированные с помощью функции `wp_schedule_event()`, действительно будут выполняться, когда они назначены. Это почти правда.

В системах Unix служба Cron запускается каждую минуту (как правило), чтобы проверить, есть ли сценарий для запуска. В WordPress эта проверка выполняется при каждой загрузке страницы. Поэтому если никто не загружает ваш веб-сайт в определенный день или загружаются только страницы из статического кэша, то ваши задания Cron могут не запускаться в этот день. Они будут срабатывать при загрузке следующей страницы.

Эта настройка подходит для обычных сайтов WordPress, но нашим приложениям нужна надежность. К счастью, легко отключить встроенный таймер Cron и настроить его на веб-сервере, чтобы он срабатывал тогда, когда это необходимо.

Чтобы отключить таймер Cron WordPress, просто добавьте следующую строчку в ваш файл `wp-config.php`:

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

Вы по-прежнему добавляете и управляете событиями, как мы делали ранее; эта константа включает или отключает *проверку* для событий, которые готовы к запуску. Нам просто нужно вручную обращаться к файлу `wp-cron.php` в нашей установке WordPress достаточно часто, чтобы при необходимости запускать наши сценарии.

Если у вас есть только ежедневные сценарии, то можно добавить задачу Cron, например, через команду `crontab -e`:

```
0 0 * * * wget -O - -q -t 1 http://yoursite.com/wp-cron.php?doing_wp_cron=1.
```



Информацию о том, как использовать Cron, можно найти на его странице вики (bwwwp.com/cronwiki). Обратитесь к руководству `wget` (bit.ly/gnu-manual), чтобы узнать, как применять `wget`.

Часть `0 0 * * *` предыдущей записи указывает Cron выполнять этот скрипт в 0 минут в 0 часов (полночь) каждый день недели.

Часть `wget -O - -q -t 1 http://yoursite.com/wp-cron.php?doing_wp_cron=1` включает команду `wget` для загрузки страницы `wp-cron.php` в вашей установке WordPress. Ключ `-O` указывает `wget` отправлять вывод в `devnull`, а `-q` включает "тихий" режим. Это не позволит Cron добавлять файлы на ваш сервер или отправлять вам по электронной почте результаты каждого своего запуска. Ключ `-t 1` говорит Cron пробовать один раз. Это предотвратит многократное попадание `wget` на ваш сервер, если первая попытка не удалась. Если вызов `wp-cron.php` не удался, возможно, остальная часть вашего сайта тоже не работает; надеюсь, вы уже были уведомлены.

Обязательно измените `yoursite.com` на фактический URL вашего сайта. И наконец, условие `?doing_wp_cron=1` в конце URL-адреса необходимо, поскольку `wp-cron.php` проверит этот параметр `$_GET` перед запуском.



Дважды проверьте, что URL-адрес `wp-cron.php` исключен из любых механизмов кэширования, которые вы, возможно, установили на своем сайте.

Одно задание Cron будет запускаться каждый день, как и любое ежедневное задание Cron, запланированное вами в WordPress, будет запускаться ежедневно. Если вам нужно, чтобы ваши задания запускались чаще, можно изменить запись Cron так, чтобы он запускался каждый час или каждые несколько минут. Обратите внимание, что вызов `wp-cron.php` по сути является обращением к нашему сайту. Проверка каждую минуту фактически то же самое, что и 1440 пользователей, посетивших ваш сайт. Так что планируйте свои задания Cron консервативно.

Использование только серверного Cron

Если вы не распространяете свой код или не хотите сообщать своим пользователям, что они должны устанавливать задания Cron на стороне сервера, вам вообще не нужно планировать свои события Cron в WordPress. Просто запланируйте работу Cron на стороне сервера, которая вызывает специальный URL-адрес для запуска вашего обратного вызова. Это особенно полезно, если вам нужно больше контролировать время работы ваших заданий или иначе удобнее управлять заданиями в Unix, а не в WordPress.



Информация о планировании серверных заданий Cron в этом разделе может использоваться для замены WP-Cron для повторяющихся событий. Отдельные события, установленные с помощью функции `wp_schedule_single_event()`, все равно должны быть обработаны с использованием WP-Cron или другого механизма.

Если бы мы выполняли задание Cron по понедельникам ранее, то обновили бы код в WordPress, как показано в следующем примере.

Пример

```
// Запускать по понедельникам
function sp_monday_cron()
{
```

```

        // проверяем, что параметр cron был передан
        if(empty($_REQUEST['sp_cron_monday']))
            return false;

        // выполнять этот код по понедельникам
    }
    add_action("init", "sp_monday_cron");

```

А ваша запись Cron будет выглядеть так:

```
0 0 * * 1 wget -O - -q -t 1 http://yoursite.com/?sp_cron_monday=1
```



И снова, еще раз проверьте, что URL-адрес `?sp_cron_monday=1` исключен из любых механизмов кэширования, которые вы, возможно, установили на своем сайте.

Функция *WP Mail*

Функция `wp_mail()` заменяет встроенную в PHP функцию `mail()`. Выглядит она примерно так:

```
wp_mail($to, $subject, $message, $headers, $attachments)
```

Ее атрибуты следующие:

- ◆ `$to` — адрес электронной почты, разделенный запятыми список адресов электронной почты или массив адресов электронной почты, на которые будет отправлено письмо (с помощью поля "To:").
- ◆ `$subject` — тема письма.
- ◆ `$message` — тело письма. По умолчанию электронное письмо отправляется как текстовое сообщение и не должно содержать HTML. Однако если вы измените тип контента (см. следующий пример), то можете включить HTML-код в ваше сообщение.
- ◆ `$headers` — необязательный массив почтовых заголовков для отправки вместе с сообщением. Это может быть использовано для добавления CC, BCC и других расширенных заголовков почты.
- ◆ `$attachments` — имя файла или массив имен файлов, которые будут прикреплены к исходящей электронной почте.

Есть два основных преимущества `wp_mail()` по сравнению с `mail()`:

- ◆ Функция `wp_mail()` может использоваться с хуками. Фильтр `wp_mail` передаст массив всех параметров, переданных в функцию `wp_mail()` для фильтрации. Вы также можете отфильтровать адрес отправки, с помощью фильтров `wp_mail_from` и `wp_mail_from_name`.
- ◆ Функция `wp_mail()` может передавать одно имя файла или массив имен файлов в параметрах `$attachments`, которые будут прикреплены к исходящей электронной

почте. Присоединение файлов к электронным письмам очень сложно, но `wp_mail()` облегчает это, обращившись вокруг класса `PHPMailer`, который сам обрабатывает функцию `PHP mail()` по умолчанию.

Отправка более приятных писем с помощью WordPress

По умолчанию электронные письма, формируемые с помощью функции `wp_mail()`, отправляются с адреса администратора, установленного на странице "General Settings" панели администратора, с указанием имени "WordPress". Это не идеально. Вы можете изменить эти значения, посредством фильтров `wp_mail_from` и `wp_mail_from_name`.

Также по умолчанию электронные письма отправляются с содержимым в виде простого текста. Вы можете применить фильтр `wp_mail_content_type` для отправки ваших электронных писем с использованием HTML.

Наконец, приятно добавлять стилизованный верхний и нижний колонтитулы ко всем исходящим письмам. Вы можете сделать это, отфильтровав сообщение электронной почты с помощью фильтра `wp_email`.

В следующем примере эти методы объединены, чтобы привести в порядок электронные письма, отправляемые вашим приложением WordPress.

Пример

```
// Обновление полей адреса электронной почты и имени отправителя
function sp_wp_mail_from($from_email)
{
    return 'info@schoolpress.me';
}

function sp_wp_mail_from_name($from_name)
{
    return 'SchoolPress';
}

add_filter('wp_mail_from', 'sp_wp_mail_from');
add_filter('wp_mail_from_name', 'sp_wp_mail_from_name');

// отправляем электронные письма в формате HTML вместо простого текста
function sp_wp_mail_content_type($content_type)
{
    if($content_type == 'text/plain')
    {
        $content_type = 'text/html';
    }
    return $content_type;
}

add_filter('wp_mail_content_type', 'sp_wp_mail_content_type');
```



```
// добавляем верхний и нижний колонтитулы из файлов в функцию активной темы
function sp_wp_mail_header_footer($email)
{
    // получаем заголовок
    $headerfile = get_stylesheet_directory() . "email_header.html";
    if(file_exists($headerfile))
        $header = file_get_contents($headerfile);
    else
        $header = "";

    // получить подвал
    $footerfile = get_stylesheet_directory() . "email_footer.html";
    if(file_exists($footerfile))
        $footer = file_get_contents($footerfile);
    else
        $footer = "";

    // обновить сообщение
    $email['message'] = $header . $email['message'] . $footer;

    return $email;
}
add_filter('wp_mail', 'sp_wp_mail_header_footer');
```

Отправка электронных писем с вашего сервера может вызвать некоторые проблемы в сети. Запуск локального SMTP-сервера для отправки электронных писем может занимать много времени в дополнение к работе веб-сервера. На доставку ваших писем могут повлиять спам-фильтры, которые не вошли в белый список IP-адресов ваших приложений. Плагин Configure SMTP (bwwawp.com/config-smtp) можно использовать для отправки исходящей электронной почты через внешний SMTP-сервер, такой как учетная запись Google Apps. Такие сервисы, как Mandrill и SendGrid, каждый из которых имеет свой собственный плагин WordPress, также предлагают способы отправки электронной почты с их доверенных серверов с дополнительным отслеживанием коэффициентов открытия и отказов.

API заголовка файла

Блок комментариев в верхней части файлов основной темы и плагинов часто называют *заголовком*. API заголовка файла состоит из трех функций, `get_plugin_data()`, `wp_get_theme()` и `get_file_data()`, которые позволяют вам анализировать эти блоки комментариев.

В следующем примере, как напоминание, показано, как может выглядеть заголовок файла плагина.

Пример

```
/*
Plugin Name: Paid Memberships Pro
Plugin URI: http://www.paidmembershipspro.com
Description: Plugin to Handle Memberships
Version: 1.7.3.2
Author: Stranger Studios
Author URI: http://www.strangerstudios.com
*/
```

Вы можете получить эти данные в массив, вызвав функцию `get_plugin_data()`:

```
get_plugin_data($plugin_file, $markup = true, $translate = true).
```

Ее атрибуты:

- ◆ `$plugin_file` — абсолютный путь к основному файлу плагина, где будет проанализирован заголовок;
- ◆ `$markup` — флаг, который при значении `true` будет применять HTML-разметку к некоторым значениям заголовка. Например, URI плагина будет превращен в ссылку;
- ◆ `$translate` — флаг, который, если установлен в `true`, будет переводить значения заголовка с использованием текущей локализации и области.

В следующем примере код перебирает каталог плагинов и отображает данные для большинства плагинов. На самом деле для того, чтобы найти все форматы всех плагинов, требуется не так много логики. Для этого вполне подойдет функция `get_plugins()`, которая будет возвращать массив всех плагинов, либо вы можете посмотреть на код этой функции, он находится в `wp-admin/includes/plugin.php`. Дополнительную информацию о `get_plugins()` можно найти в Кодексе WordPress на странице, посвященной функциям (b2wawp.com/funct-ref).

Пример

```
// Код должен включать этот файл
require_once(ABSPATH . "wp-admin/includes/plugin.php");

// запоминаем текущий каталог
$cwd = getcwd();

// переключаемся в каталог тем
$plugins_dir = ABSPATH . "wp-content/plugins";
chdir($plugins_dir);

echo "<pre>";

// перебираем каталоги тем и печатаем информацию о теме
foreach(glob("*", GLOB_ONLYDIR) as $dir)
```

```

{
    $plugin = get_plugin_data($plugins_dir .
    "/" . $dir . "/" . $dir . ".php", false, false);
    print_r($plugin);
}

echo "</pre>";

// переключиться обратно в текущий каталог на всякий случай
chdir($cwd);

```

Точно так же вы можете, вызвав `wp_get_theme()`, получить информацию из заголовка файла темы:

```
wp_get_theme($stylesheet, $theme_root)+
```

Атрибуты следующие:

- ◆ `$stylesheet` — название каталога для темы. Если не установлено, этот параметр будет каталогом текущей темы;
- ◆ `$theme_root` — абсолютный путь к корневой папке темы. Если не установлено, используется значение, возвращаемое функцией `get_raw_theme_root()`.

В следующем примере код перебирает каталог тем и отображает данные для *большинства* тем. На самом деле, чтобы найти все темы, требуется тоже не так много логики. Для этого подойдет функция `get_themes()`, которая будет возвращать массив всех объектов `WP Theme`. Код для этой функции находится в файле `wp-admin/includes/theme.php`. Дополнительную информацию о `wp_get_themes()` можно найти на их странице Кодекса WordPress (bwwwp.com/wp-get-theme).

Пример

```

// Запоминаем текущий каталог
$cwd = getcwd();

// переключаемся в каталог тем
$themes_dir = dirname(get_template_directory());
chdir($themes_dir);

echo "<pre>";

// перебираем каталоги тем и печатаем информацию о теме
foreach(glob("*", GLOB_ONLYDIR) as $dir)
{
    $theme = wp_get_theme($dir);
    print_r($theme);
}

```

```
echo "</pre>";
```

```
// переключиться обратно в текущий каталог на всякий случай  
chdir($cwd);
```

Добавление заголовков файлов в ваши собственные файлы

Функции `get_plugin_info()` и `wp_get_theme()` используют функцию `get_file_data()`. Вы можете получить доступ к функции `get_file_data()` напрямую, чтобы извлечь заголовки файлов из любого файла. Это может помочь вам создать ваши собственные плагины или суб-плагины (часто называемые *модулями* или *дополнениями*) для ваших плагинов.

Атрибуты функции `get_file_data($file, $default_headers, $context = "")`:

- ◆ `$file` — полный путь и имя файла, из которого нужно извлечь данные;
- ◆ `$default_headers` — массив полей заголовка для поиска. Ключами массива должны быть имена заголовков, а значениями массива должны быть регулярные выражения извлечения меток, которые стоят перед : в комментарии. Обычно вы можете просто ввести имя заголовка в качестве регулярного выражения;
- ◆ `$context` — метка для различения разных типов заголовков. Этот параметр определяет, какой фильтр `extra_{context}_headers` применяется по умолчанию к переданным заголовкам, как показано в следующем примере.

Пример

```
// Устанавливаем заголовки для наших файлов  
$default_headers = array(  
    "Title" => "Title",  
    "Slug" => "Slug",  
    "Version" => "Version"  
);  
  
// запоминаем текущий каталог  
$cwd = getcwd();  
  
// перейти в каталог отчетов  
$reports_dir = dirname(__FILE__) . "/reports";  
chdir($reports_dir);  
  
echo "<pre>";  
  
// цикл по .php файлам в каталоге отчетов  
foreach (glob("*.php") as $filename)
```

```
{
    $data = get_file_data($filename, $default_headers, "report");
    print_r($data);
}

echo "</pre>";

// возвращаемся к текущему каталогу, если кто-то далее ожидает путь по умолчанию
chdir($cwd);
```

Добавление новых заголовков в плагины и темы

Фильтры `extra_plugin_headers` и `extra_theme_headers` позволяют добавлять другие заголовки к информации, возвращаемой `get_plugin_data()` и `wp_get_theme()` соответственно. Вы также можете использовать фильтры для своих пользовательских контекстов, чтобы работать с `get_file_data()`. Формат — `extra_{context}_headers`.

В листинге 7.4 показано добавление заголовка `Allow Updates` к плагинам. Если этот заголовок найден, а значение равно `no` или `false`, то данный плагин не будет включен в список для обновления.

Листинг 7.4. Плагин Stop Plugin Updates

```
<?php
/*
Plugin Name: Stop Plugin Updates
Plugin URI: http://bwawwp.com/plugins/stop-plugin-updates/
Description: "Allow Updates: No" i a plugin's header keeps it from updating.
Version: .1
Author: Stranger Studios
Author URI: http://www.strangerstudios.com
*/

// добавляем заголовок AllowUpdates в плагин
function spu_extra_plugin_headers($headers) {
    $headers['AllowUpdates'] = "Allow Updates";
    return $headers;
}
add_filter("extra_plugin_headers", "spu_extra_plugin_headers");

/*
    проходим циклом по плагинам
    проверяем, запрещены ли обновления, и если да, удалить его из списка
*/
function spu_pre_set_site_transient_update_plugins($update_plugins) {
    // посмотреть, есть ли плагины, требующие обновления
    if (!empty($update_plugins) && !empty($update_plugins->response)) {
```

```
// проходим циклом по плагинам
$new_plugins = array();
foreach ($update_plugins->response as $pluginpath => $plugin) {
    // проверяем, разрешен ли плагин или нет
    $plugin_data = ABSPATH . '/wp-content/plugins/' . $pluginpath;
    $plugin_data = get_plugin_data($plugin_data);
    if (strtolower($plugin_data['Allow Updates']) == "no" ||
        strtolower($plugin_data['Allow Updates']) == "false") {
        // изменить проверенную версию и не добавлять в ответ
        $update_plugins->checked[$pluginpath] = $plugin_data['Version'];
    }
    else {
        // не заблокирован. добавить плагин к новому ответу
        $new_plugins[$pluginpath] = $plugin;
    }
}
$update_plugins->response = $new_plugins;
}

return $update_plugins;
}

add_action(
    'pre_set_site_transient_update_plugins',
    'spu_pre_set_site_transient_update_plugins'
);
?>
```

Heartbeat API

API Heartbeat, представленный в WordPress 3.9, представляет собой почти AJAX-коммуникацию практически в реальном времени для передачи и извлечения определенного контента из базы данных WordPress и обратно. Если активен, API Heartbeat по умолчанию будет выполняться каждые 15–30 секунд на загруженной странице, чтобы проверить, произошло ли что-то; это называется *импульсом Heartbeat*. Вы также можете установить импульс вручную в коде программы на срок до 60 секунд.



Ознакомьтесь с плагином *Heartbeat Control*, разработанным Джеффом Матсоном. Он отлично подходит для простой настройки того, как часто импульсы Heartbeat отправляются на сервер и откуда посылаются импульсы Heartbeat. Вы даже можете отключить их.

API Heartbeat — это скорее бэкэнд-функция WordPress, и хотя его можно использовать и на фронтенде, так, вероятно, не следует делать (или, по крайней мере, следует придерживаться принципа экономии). Как вы можете себе представить, API Heartbeat может быть довольно тяжелым для серверных ресурсов, в зависимости от

цели применения и числа пользователей. В ядре WordPress этот API служит для таких операций, как блокировка постов. Вы знаете это милое административное всплывающее окно на редактируемом вами посте, которое дает вам знать, что кто-то еще хочет отредактировать этот пост? *Если это случится с вами, вы могли бы подумать: разве они не видят маленькую иконку замка рядом с сообщением и всплывающее предупреждение, говорящее, что я сейчас над этим работаю?* Это отличный пример использования Heartbeat API для проверки статуса того, кто работает над сообщением, и принятия соответствующих мер. Когда страница списка публикаций загружается впервые, известно, кто на каких страницах работает; затем API Heartbeat "вступает во владение", чтобы проверять каждые *x* секунд, закончили ли пользователи публикации, над которыми они работали, или кто-то открыл другое сообщение и начал работать над ним. WordPress получает эти действия с загруженной страницы, отправляя данные асинхронно через API Heartbeat на сервер, который будет обработан и возвращен для обновления загруженной страницы через JavaScript. Таким образом, этот человек точно знал, что вы работали над данным постом, и решил все равно взять его на себя.

Работа API Heartbeat включает три основных этапа:

1. Отправка данных на сервер с клиента через jQuery. Это делается с помощью события `heartbeat-send`.
2. Обработка запроса на сервере и предоставление ответа через PHP. Это осуществляется с помощью фильтра `heartbeat_received`.
3. Обработка ответа на клиенте через jQuery. Это делается с помощью события `heartbeat-tick`.

В нашем примере мы создадим простой плагин, который использует API Heartbeat для уведомления сотрудников начальной школы, когда родители приезжают, чтобы забрать своих детей.

Практически во всех начальных школах, с которыми мы сталкиваемся, есть этот архаичный и опасный метод, когда учителя идут к машине в жару, холод, дождь и снег, чтобы родители подписали лист бумаги, чтобы забрать своих детей. Но, уважаемые члены школьного совета, в 2020 году мы можем сделать что-то лучше!

Сначала давайте создадим базовый плагин, который отображает экран администратора на мониторе в кафетерии, где ученики терпеливо ждут прибытия своих родителей. Это будет сокращенная версия концепции плагина WordPress, короткая и приятная, чтобы подчеркнуть функциональность Heartbeat API. Выполните следующие шаги.

1. Создайте в каталоге плагинов папку с именем `the-teacher-life-saver`.
2. Создайте в этой папке файл с именем `the-teacher-life-saver.php`.
3. Проанализируйте и скопируйте код листинга 7.5.

Сохраните и активируйте плагин, затем протестируйте его на панели администратора.

Программа, приведенная в листинге 7.5, создает виджет панели администратора, который автоматически отображает контент, обработанный на сервере через API-интерфейс WP Heartbeat, без обновления страницы вручную.

Листинг 7.5. Плагин Teacher Life Saver

```
<?php
/**
 * Plugin Name: The Teacher Life Saver
 * Plugin URI:  https://schoolpress.me/
 * Description: Оповещение учителей о прибытии родителей
 * Version:     0.0.1
 */

// Хук для создания функции виджета сводной панели
function ttls_dashboard_widget() {
    global $wp_meta_boxes;

    // идентификатор виджета, имя виджета, функция обратного вызова
    wp_add_dashboard_widget('ttls_widget', 'Student Pick Ups', 'ttls_dashboard');
}

add_action('wp_dashboard_setup', 'ttls_dashboard_widget');

// Разметка для функции виджета сводной панели
function ttls_dashboard() {
    echo "<div id='ttls_message'></div>";
}

// Изменяем импульс по умолчанию на каждые 15 секунд,
// чтобы нам не пришлось долго ждать функция
function ttls_heartbeat_settings($settings) {
    $settings['interval'] = 15; // Что-нибудь между 15-60 секундами
    return $settings;
}

add_filter('heartbeat_settings', 'ttls_heartbeat_settings', 1);

// применяем heartbeat.js и нашу функцию JavaScript
function ttls_heartbeat_init()
{
    // Запускаем это только на странице панели инструментов
    global $pagenow;
    if($pagenow != 'index.php')
        return;

    // Применяем API Heartbeat
    wp_enqueue_script('heartbeat');
```



```

// Загружаем наши функции JavaScript в подвал
add_action("admin_footer", "ttls_js_wp_footer");
}

add_action("admin_init", "ttls_heartbeat_init");

// Функции JavaScript выполнены в подвале
function ttls_js_wp_footer()
{
?>
<script>
jQuery(document).ready(function() {

// Используем heartbeat-send для отправки любых ключей/значений в массиве данных
jQuery(document).on('heartbeat-send', function(e, data) {
    data['client'] = 'check-for-parents';
});

// Используем функцию heartbeat-tick для проверки данных и выполняем действие
jQuery(document).on('heartbeat-tick', function(e, data) {
if(data['server'])
    document.getElementById("ttls_message").innerHTML = data['server']
    + document.getElementById("ttls_message").innerHTML;
});
});
</script>
<?php
}

// Серверная функция для получения и обработки запроса,
// которая затем он возвращает ответ
function ttls_heartbeat_received($response, $data)
{
// Проверяем, какие данные были переданы из функции JS heartbeat-send
if($data['client'] == 'check-for-parents')
{
    // Создать любой ответ, который вы хотите
    $r = '<p>';
    $r .= date('m/j/y g:i a', current_time('timestamp', 0));
    $r .= " - Nina Messenlehner's father Brian Messenlehner ";
    $r .= "has arrived in a 2007 White Hummer H2.";
    $r .= '</p>';

    return $r;
}
}

add_filter('heartbeat_received', 'ttls_heartbeat_received', 10, 2);
?>

```

Если вы сделали все правильно, то должны увидеть, что любой контент, который вы поместили в переменную `$response[server]` в функции `ttls_heartbeat_received()`, появляется в вашем новом виджете сводной панели каждые 15 секунд. Вы действительно можете выполнить любой процесс, который хотите на стороне сервера, и вернуть все, что вы хотите, в ответе, который будет обработан на стороне клиента. В нашем примере мы в основном жестко запрограммировали простой ответ, чтобы продемонстрировать, как он работает, но вы можете пойти еще дальше. Представьте, что у родителя есть мобильное приложение на смартфоне с геолокацией вокруг школьной парковки. Как только родитель выезжает на парковку, статус может быть обновлен; и в следующий раз, когда импульс Heartbeat проверяет наличие новых родителей, он может вывести информацию об этом родителе на монитор. В нашем примере мы вернули лишь основную информацию, но мы могли бы включить гораздо больше: например, имя ученика, имя родителя, фотографию родителя, тип и фотографию транспортного средства, номерной знак и все остальное, что может быть важным.

Безопасность в WordPress

Поберегитесь, хакеры! В этой главе будет представлено множество советов и рекомендаций о том, как сделать сайты на базе WordPress более безопасными и не дать им пасть жертвой злоумышленных намерений.

Почему это важно

Вне зависимости от размеров создаваемого вами сайта, вам ни в коем случае не следует забывать о его безопасности. Сайт любого размера может стать жертвой хакеров или вредоносных программ. Хорошая осведомленность в плане безопасности сайтов на базе WordPress и принятие соответствующих превентивных мер поможет вам снизить уязвимость к атакам, а возможно, и полностью их избежать.

Одна из самых популярных разновидностей атак — *атака методом прямого перебора*, при которой бот или скрипт пытается получить доступ к сайту путем подбора правильной комбинации из имени пользователя и пароля. Хотя, на первый взгляд, это не кажется чем-то опасным, следует иметь в виду, что такие боты часто представляют собой огромные сети компьютеров, способные проверять сотни и даже тысячи вариантов за одну секунду! Даже если этим ботам не удастся получить доступ к панели администратора WordPress, они вполне могут вывести ваш сайт из строя просто за счет того, что реагирование на вредоносные запросы будет отнимать у сервера огромное количество ресурсов. В такое состояние "*отказа в обслуживании*" (Denial of Service, DoS) сайт может привести целенаправленная атака, атака методом прямого перебора или воздействие автоматических спамеров.

В данной главе мы рассмотрим возможности в плане обеспечения безопасности, встроенные в стандартный вариант установки WordPress, а также ряд других легких в применении советов по повышению безопасности сайта. Мы также коснемся того, какие плагины могут помочь с решением таких дополнительных проблем, как спам.

Если вы решите *не* читать дальнейшее содержание этой главы, то с вами может произойти одна из не очень приятных ситуаций. Вот лишь некоторые наиболее распространенные сценарии:

- ◆ Вы попытаетесь ввести адрес своего сайта, но обнаруживаете, что он не работает. А сайт ведь не должен простаивать! Надо надеяться, что у вас будет резервная копия и вы сможете быстро восстановить его работоспособность.
- ◆ Вы заметили, что ссылки на ваш сайт начинают отображаться в результатах поиска, выдаваемых при поиске "Виагры" и других препаратов для повышения мужской потенции. Это может плохо отразиться на вашем бизнесе, если ваш сайт не предназначен исключительно для продажи таких препаратов.

- ◆ Ваше приложение рассылает всем вашим сотрудникам электронные письма со ссылкой для загрузки компьютерного вируса, в чем они абсолютно не нуждаются!
- ◆ Ваше приложение было взломано, в результате чего были раскрыты персональные данные ваших сотрудников (их имена, адреса, номера телефонов и адреса электронной почты).
- ◆ Ваш сайт был взломан и использован для заражения других сайтов вредоносным программным обеспечением. А это самый быстрый способ оказаться исключенным из списка результатов поиска Google и других важных каталогов.

Основные меры безопасности

Далее изложены весьма простые, но в то же время и наиболее важные советы в плане безопасности. Обратите на них самое пристальное внимание, и это сэкономит вам массу времени, денег и нервов ваших посетителей или сотрудников.

Регулярно выполняйте обновление

Первый и самый важный момент в плане обеспечения безопасности состоит в том, чтобы обновляться до самой последней версии WordPress сразу после ее выпуска, а также постоянно обновлять все установленные на сайте плагины/темы. Поскольку обновления часто включают в себя улучшения в плане безопасности, чтобы ваше ПО оставалось актуальным и безопасным, важно постоянно его обновлять.

Не используйте имя пользователя "admin"

Следующий важный момент состоит в том, чтобы имя пользователя "admin" не использовалось для входа ни в одну из учетных записей ваших пользователей. Многие боты автоматически пытаются авторизоваться на сайте через это имя пользователя. Поскольку, как известно, пользователи очень редко меняют этот логин на что-то другое, задача ботов сильно упрощается — им остается лишь подобрать пароль. При установке WordPress пользователю по умолчанию предоставляется имя "admin", и, чтобы этого не произошло, вы *должны* специально выбрать другое имя! Если вы уже задали имя пользователя "admin" для входа в WordPress, то создайте новую учетную запись пользователя с правами администратора, авторизуйтесь под этим новым логином и удалите учетную запись администратора, предоставленную по умолчанию. *Не забудьте соответственно изменить принадлежность всех постов и страниц, созданных учетной записью администратора.*

Выбирайте надежный пароль

Также важно выбирать надежный пароль, особенно для учетных записей с правами администратора. Не используйте какое-либо слово или имя целиком. Вместо этого лучше задать случайное сочетание символов, не связанное с вами лично.

В пароле должны присутствовать и прописные, и строчные буквы, а также цифры и специальные символы. Хороший пароль должен включать в себя не менее 10 символов; чем больше символов в пароле, тем он надежнее. Если вы не можете придумать себе пароль, просто беспорядочно понажимайте клавиши своей клавиатуры или воспользуйтесь таким сервисом, как Random.org (www.random.org/passwords/). Как следует запомните созданный пароль или скопируйте его куда-нибудь и должным образом сохраните. WordPress сообщит вам о том, насколько надежен ваш пароль, отнеситесь к этой информации с должным вниманием.

Примеры плохих паролей

Вот несколько примеров плохих паролей:

- ◆ password;
- ◆ password123;
- ◆ pa55w0rd;
- ◆ 123456789;
- ◆ qwerty;
- ◆ batman;
- ◆ mustang;
- ◆ letmein.

Нецелесообразно использование в качестве пароля любой вариации отдельных слов, цифр или имен:

- ◆ usmarine (Брайан служил в морской пехоте);
- ◆ brianmessenlehner (имя и фамилия Брайана);
- ◆ jason&kim050507 (имя Джейсона, имя его жены и дата их свадьбы);
- ◆ Dalya-Brian (имена детей);
- ◆ ThaiShortiMaxx (имена домашних животных);
- ◆ IMAWESOME! ("Я ОФИГЕННЫЙ!" — такие фразы легко подобрать, поскольку они широко известны).

Такой пароль, теоретически, может подобрать любой, кто хоть немного знаком с пользователем или членами его семьи.

Примеры хороших паролей

Вот несколько примеров надежных паролей:

- ◆ U\$s(#8H27@!
- ◆ !lik32EaTF1\$h&CHIp5
- ◆ #Uk@nN0tBr3akTh1s\$h1t!!!
- ◆ [0mG-LoL-R0F1-T0T3\$CraY]!

Какой бы раздражающей ни казалась вам необходимость потратить пару лишних секунд на выдумывание хорошего пароля, эти усилия с лихвой окупят себя, когда защитят ваш сайт/приложение от взлома.

Усиление защиты в WordPress

Принятие мер по повышению безопасности сайта часто называют *"усилением защиты"*. В разделе "Усиление защиты WordPress" справочной системы WordPress (oreil.ly/vTDa-) рассматривается все, о чем говорится в данной главе, а также ряд моментов, которых мы здесь не касаемся. Рекомендуется сначала прочитать данную главу, а затем обратиться к этому разделу справочной системы за дополнительной информацией.

Теперь давайте рассмотрим ряд способов, позволяющих усложнить взлом вашего приложения.

Запретите администраторам редактировать плагины и темы

По умолчанию WordPress предоставляет администраторам возможность редактировать исходный код любого плагина или темы непосредственно в веб-браузере. Вам следует отключить эту функциональность, чтобы в том случае, если хакеру удастся войти в один из администраторских аккаунтов, он не мог добавить вредоносный код через пользовательский интерфейс администратора для редактирования кода. Чтобы отключить эту функциональность, добавьте следующую строку в файл `wp-config.php`:

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

Измените префикс таблиц базы данных

В стандартном варианте установки WordPress для всех таблиц базы данных задан префикс `wp_`. Просто изменив этот префикс на что-либо иное, вы сделаете свой сайт гораздо менее уязвимым для хакеров, пытающихся внедрить SQL-код, предполагая, что у вас обычный префикс `wp_`. Сразу после установки WordPress у вас есть возможность выбрать для таблиц любой нужный вам префикс; вместо предлагаемого по умолчанию префикса `wp_` задайте какой-либо кастомный префикс.

В случае уже работающего сайта на базе WordPress для этого потребуется выполнить следующие шаги:

- ♦ Создайте резервную копию базы данных на случай неудачного завершения данной операции.
- ♦ Откройте файл `wp-config.php` и измените строку `$table_prefix = wp_;` на `$table_prefix = anyprefix_;`.

- ◆ Обновите имена имеющихся таблиц базы данных так, чтобы они включали в себя новый префикс, выполнив приведенные в следующем примере команды языка SQL с помощью приложения PhpMyAdmin или любого другого SQL-клиента наподобие MySQL Workbench.

Пример

```
rename table wp_commentmeta to anyprefix_commentmeta;  
rename table wp_comments to anyprefix_comments;  
rename table wp_links to anyprefix_links;  
rename table wp_options to anyprefix_options;  
rename table wp_postmeta to anyprefix_postmeta;  
rename table wp_posts to anyprefix_posts;  
rename table wp_terms to anyprefix_terms;  
rename table wp_term_relationships to anyprefix_term_relationships;  
rename table wp_term_taxonomy to anyprefix_term_taxonomy;  
rename table wp_usermeta to anyprefix_usermeta;  
rename table wp_users to anyprefix_users;
```



Также следует выполнить аналогичные SQL-запросы `rename` для всех кастомных таблиц, добавленных вашим приложением или используемыми плагинами.

Используя SQL-команды или SQL-клиент, обновите таблицы `anyprefix_options` и `anyprefix_usermeta`, заменив все вхождения префикса `wp_` на префикс `anyprefix_`:

```
update anyprefix_options set option_name = replace(  
option_name, 'wp_', 'anyprefix_');  
update anyprefix_usermeta set meta_key = replace(  
meta_key, 'wp_', 'anyprefix_');
```

Протестируйте свой сайт и убедитесь, что все работает должным образом.

Если ваших навыков не хватает для того, чтобы уверенно внести эти изменения вручную, вы можете изменить префикс таблиц с помощью следующих плагинов:

- ◆ Change Table Prefix (bwawwp.com/change-tp/);
- ◆ Brozzme DB Prefix & Tools Addons (oreil.ly/qhvl1m).

Переместите в другое место файл `wp-config.php`

В файле WordPress `wp-config.php` содержится такая ценная информация, как путь к вашей базе данных, имя пользователя, пароль и ключи аутентификации WordPress. Эти данные сохраняются в переменных языка PHP и не предоставляются браузеру, что снижает вероятность несанкционированного доступа к ним, но не исключает такой возможности полностью. Вы можете переместить файл `wp-config.php` на один уровень вверх по отношению к каталогу установки WordPress; в общем случае это должен быть непубличный каталог. Не найдя файл `wp-config.php` в корневом катало-

ге, WordPress автоматически выполнит поиск в каталоге, расположенном выше. Например, можно переместить файл `wp-config.php` из каталога `/username/public_html/` в каталог `/username/`.

Также можете сохранить `wp-config.php` в виде файла с любым именем в любом каталоге. Для этого создайте копию файла `wp-config.php` и присвойте ей нужное имя, после чего переместите ее в любой каталог, расположенный выше корневого каталога установки WordPress. В своем исходном файле `wp-config.php` удалите весь код и добавьте инструкцию `include` с относительным путем и именем файла созданной копии. Например, можно скопировать файл `/username/public_html/wp-config.php` в файл `/username/someotherfolder/stuff.php`, изменив код в файле `wp-config.php` на `include('/username/someotherfolder/stuff.php');`.

Не отображайте сообщения об ошибках авторизации

Обычно WordPress отображает сообщение об ошибке в том случае, если, авторизуясь на сайте, вы вводите неправильный логин или пароль. К сожалению, это дает хакерам возможность узнать, что именно они делают неправильно, когда пытаются получить доступ к сайту.

К счастью, эту проблему можно легко исправить — нужно лишь добавить в файле темы `functions.php` или в кастомном плагине строку кода, скрывающую или изменяющую эти сообщения:

```
add_filter('login_errors', function ($message) {  
    return "Invalid username or password.";  
});
```



В представленном коде используется анонимная функция (oreil.ly/3emCk) в качестве обратного вызова во втором параметре вызова `add_filter()`. Для этого необходим PHP версии 5.3 или выше. Вы также можете просто определить именованную функцию до вызова `add_filter()`, но тогда это уже не будет "одной строкой" кода.

Не отображайте номер версии WordPress

Многие боты целенаправленно ищут в Интернете сайты на базе WordPress определенных версий, с тем, чтобы использовать известные уязвимости этих версий. По умолчанию WordPress помещает между тегами `<head>` и `</head>` каждой страницы следующий код:

```
<meta name="generator" content="WordPress 3.8.1" />
```

Вы можете легко скрыть номер используемой версии WordPress, реализовав следующий код:

```
add_filter('the_generator', '__return_null');
```




Узнать, использует ли сайт WordPress, и, если да, то какой версии, можно, разными способами. Например, если не указана версия скрипта, можно вспомнить о том, что версия WordPress добавляется к URL-адресам JavaScript-файлов. Есть и другие, более хитроумные способы, с помощью которых хакер при желании всегда узнает номер вашей версии WordPress. Однако даже столь бесхитростное средство защиты может сыграть свою роль в предотвращении проводимых в Интернете атак, поскольку они, по большей части, носят автоматизированный характер.

Исключите возможность авторизации через страницу wp-login.php

Некоторые боты отличаются достаточно высоким "интеллектом". Только что мы обсуждали, как можно скрыть от ботов номер версии WordPress, однако зачастую боту достаточно просто убедиться в том, что вы используете WordPress. Он может легко это сделать, отправив POST-запрос к файлу wp-login.php. Убедившись в том, что файл wp-login.php существует, бот может приступить к попыткам авторизации на сайте.

Будет лучше, если мы перенаправим запросы к файлу wp-login.php на главную страницу, тем самым не дав ботам возможности осуществлять целенаправленные попытки авторизации с помощью этого файла. Выполните следующие шаги, чтобы сделать альтернативную страницу авторизации и скрыть страницу авторизации wp-login.php:

- ◆ Добавьте в свой файл *.htaccess следующее правило перезаписи:

```
RewriteRule ^new-login$ wp-login.php.
```

Обратите внимание, что для доступа к каталогу wp-admin теперь потребуется указать URL-адрес /new-login/. При желании вы можете задать и любой другой адрес.

- ◆ В своем файле темы functions.php или в кастомном плагине добавьте код, приведенный в следующем примере.

Пример

```
function schoolpress_wp_login_filter($url, $path, $orig_scheme) {
    $old = array("/(wp-login\.php)/");
    $new = array("new-login");
    return preg_replace($old, $new, $url, 1);
}

add_filter('site_url', 'schoolpress_wp_login_filter', 10, 3);

function schoolpress_wp_login_redirect() {
    if (strpos($_SERVER["REQUEST_URI"], 'new-login') === false) {
        wp_redirect(site_url());
        exit();
    }
}

add_action('login_init', 'schoolpress_wp_login_redirect');
```

- ◆ Если вы не хотите писать какой-либо кастомный код, то можете добиться аналогичного результата с помощью следующих плагинов:
 - iThemes security (bwawwp.com/themes-sec);
 - WP Admin (bwawwp.com/lockdown-wp).

Добавьте в файл *.htaccess кастомные правила, блокирующие доступ к каталогу wp-admin

Если вы единственный пользователь, которому требуется доступ к серверной части вашего приложения, или если число таких пользователей можно буквально пересчитать по пальцам, то можно ограничить доступ к серверной части определенным рядом IP-адресов. Создайте новый файл *.htaccess в каталоге wp-admin своей установки WordPress и добавьте следующий код, подставив вместо адреса 127.0.0.1 свой реальный внешний IP-адрес:

```
order deny,allow
```

```
allow from 127.0.0.1 #(продублируйте эту строку при наличии нескольких IP-адресов)
```

```
deny from all
```

Если вы не уверены в том, какой внешний IP-адрес вы используете, воспользуйтесь сайтом ipchicken.com.

Если у вас возникло подозрение, что запросы от некоторых IP-адресов отправляются ботами или злонамеренными пользователями, вы можете заблокировать эти IP-адреса, добавив следующий код:

```
order allow,deny
```

```
deny from 127.0.0.1 #(повторите эту строку при наличии нескольких IP-адресов)
```

```
allow from all
```

Если пользователь захочет обойти эту блокировку IP-адреса, он всегда сможет это сделать, воспользовавшись прокси-сервером.

Если вы ожидаете, что ваш IP-адрес или IP-адреса ваших пользователей серверной части будут часто меняться, или если у вас слишком много пользователей серверной части для того, чтобы вы могли управлять всеми их IP-адресами, то можно добавить отдельную комбинацию из имени пользователя и пароля для доступа к каталогу wp-admin. Тем самым вы получите отличный дополнительный уровень аутентификации, поскольку всем пользователям серверной части потребуется вводить имя пользователя и пароль из файла *.htaccess в дополнение к своему обычному имени пользователя и паролю для входа в WordPress:

```
AuthType Basic
```

```
AuthName "restricted area"
```

```
AuthUserFile /path/to/protected/dir/.htpasswd
```

```
require valid-user
```

Обратите внимание на строку с директивой AuthUserFile — вам нужно будет создать файл *.htpasswd в каталоге, расположенном где-либо выше или за пределами каталога установки WordPress. В этом файле вам нужно будет добавить имя поль-

зователя и пароль. При этом пароль не может представлять собой обычный текст — создайте зашифрованный пароль с помощью такого инструмента, как генератор паролей для файла *.htaccess (bwaawp.com/htaccess-pw).

К примеру, комбинация из логина и пароля `letmein/Pr3tTyPL3a$3!` после зашифровки может выглядеть как `letmein:E5Dj7cUaQVcN`.

Добавьте всю зашифрованную строку `letmein:E5Dj7cUaQVcN` в файл *.htpasswd; теперь при попытке перейти в каталог /wp-admin, у пользователей будут запрашиваться логин и пароль. Не забудьте сообщить своим пользователям серверной части, какой логин и пароль они должны вводить, и попросите их не делиться этой информацией с посторонними.

SSL-сертификаты и HTTPS

Принимая конфиденциальную информацию через веб-форму (например, номер кредитной карты), вы должны шифровать эту информацию, осуществляя загрузку и отправку формы по протоколу SSL или HTTPS. Прежде всего, следует дать несколько определений.

SSL — означает "Secure Sockets Layer" (уровень защищенных сокетов). Это технология для шифрования обмена данными с веб-страницей.

HTTP — означает "Hypertext Transfer Protocol" (протокол передачи гипертекста). Это стандартный *протокол* для обслуживания веб-страниц без шифрования.

HTTPS — означает "HTTP Secure" (защищенный протокол передачи гипертекста). Это протокол для обслуживания веб-страниц с SSL-шифрованием.

Существует много вариантов настройки протокола SSL на сервере и установки SSL-сертификата. Однако в наши дни целесообразно настраивать *все сайты* (а не только сайты электронной коммерции) на обслуживание *всего трафика* (а не только конфиденциального трафика) по протоколу HTTPS. Далее будет рассмотрено несколько вариантов такой настройки.

Установка SSL-сертификата на сервере

Прежде всего, вы должны включить протокол SSL на своем веб-сервере. Как именно вы будете это делать, зависит от того, какой хост и веб-сервер вы используете. Хостинг DigitalOcean предлагает отличную документацию по системному администрированию в целом и особенно хорошую статью о настройке SSL для сервера Apache (oreil.ly/oq7Pz).

После включения SSL-сервиса на хосте вам потребуется SSL-сертификат для его использования. Хотя самозаверенные сертификаты пригодны для целей тестирования, однако любой современный браузер выведет довольно устрашающее предупреждение, если вы попытаетесь просматривать сайт, используя самозаверенный сертификат. На рис. 8.1 показано, какое предупреждение выводит в таком случае браузер Chrome.

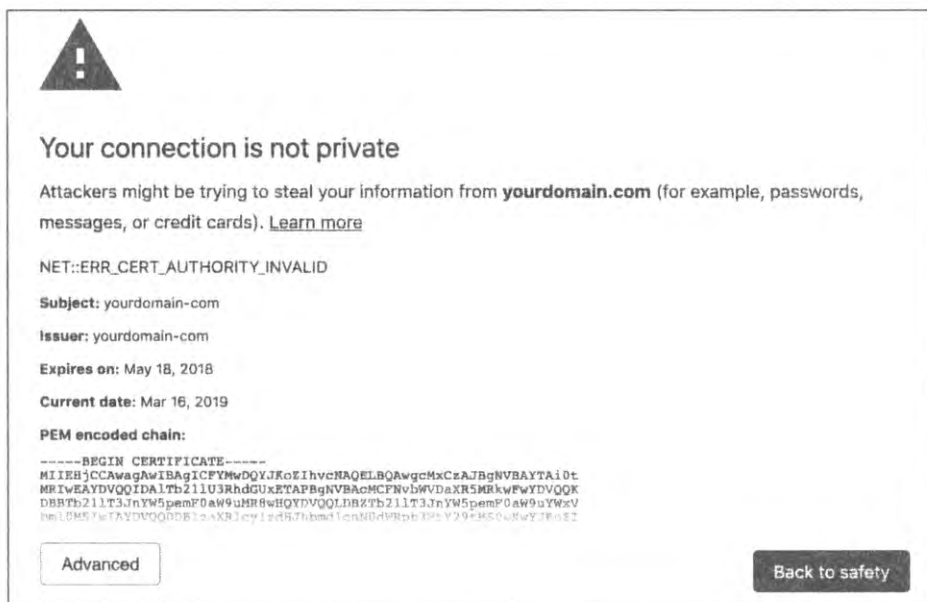


Рис. 8.1. Предупреждение браузера Chrome об использовании самозаверенного SSL-сертификата

Для эксплуатационных окружений вам потребуется сертификат с открытым ключом от некоторого центра сертификации (ЦС). Хотя сертификаты с открытым ключом можно приобретать отдельно, обычно они включаются в состав пакета хостинга или предлагаются в качестве дополнения к нему. При этом также подойдут и отдельно приобретенные SSL-сертификаты с открытым ключом. Хороший сертификат от ЦС будут считать надежным все современные веб-браузеры, на что будет указывать отображение в адресной строке значка в виде зеленого или золотистого замка, а не сломанного или красного замка, при просмотре вашего сайта.

Сегодня можно найти хорошие варианты для приобретения платных или бесплатных сертификатов от ЦС. Выбрав сертификат от ЦС, вы, фактически, подтверждаете, что домен, на котором вы его используете, действительно вам принадлежит. Право собственности на домен обычно подтверждается путем отправки электронного письма на расположенный на домене адрес. В то же время в случае центра сертификации Let's Encrypt подтверждение осуществляется путем запуска автоматизированных скриптов на рассматриваемом сервере.

Сервис Let's Encrypt

За время, прошедшее с момента публикации первого издания этой книги, в сети появился новый сервис Let's Encrypt, предлагающий легкие в установке и на 100% бесплатные сертификаты с открытым ключом. Сертификаты от сервиса Let's Encrypt устанавливаются из командной строки с помощью инструмента под названием "Certbot". Certbot доступен для большинства серверных платформ. Если ваш веб-сервер правильно настроен и подключен к Интернету, то Certbot автоматически выполнит его подтверждение в центре сертификации Let's Encrypt, после чего выдаст и установит сертификат на сервере. После этого можно будет настроить задание Cron на обновление сертификата примерно через каждые три месяца.

Сертификат от сервиса Let's Encrypt часто можно использовать на веб-хосте, даже если вы не администрируете собственный сервер. Хотя платные варианты обычно хорошо заметны на панели управления, ряд вариантов также может предлагаться на этапе настройки протокола SSL; кроме того, часто можно отправить запрос в службу поддержки с просьбой настроить ваш хост на использование бесплатного SSL-сертификата от сервиса Let's Encrypt для вашего сайта.

Повторим еще раз: на каждом сайте должен быть установлен SSL-сертификат. В первом издании этой книги было показано, как можно обслуживать страницы администратора и страницы оплаты по протоколу HTTPS, а остальную часть сайта — по незащищенному протоколу HTTP. Теперь мы уже не рекомендуем этот комбинированный вариант настройки. Интернет достиг той стадии развития, на которой предполагается, что *все веб-сайты* обслуживаются исключительно по протоколу HTTPS. Это один из элементов общей тенденции к обеспечению "безопасности по умолчанию".

Существует целый ряд доводов в пользу того, чтобы вы настроили на использование HTTPS весь свой сайт:

- ◆ Безопасность по умолчанию. Можно подумать, что в защите нуждаются страницы авторизации и оплаты, однако, что произойдет, если ваш сайт будет обновлен таким образом, чтобы форма авторизации отображалась на боковой панели? В таком случае защита потребуется уже каждой странице сайта. Если весь ваш сайт будет обслуживаться по протоколу HTTPS, то вы никогда случайно не добавите на сайт незашифрованную форму.
- ◆ Пользователи Интернета приучены обращать внимание на значок замка, показанный на рис. 8.2. Вне зависимости от уровня их опытности, они чувствуют себя увереннее, когда видят этот значок. Кроме того, современные браузеры выводят достаточно устрашающие предупреждения, если некоторая часть сайта не обслуживается по протоколу HTTPS.

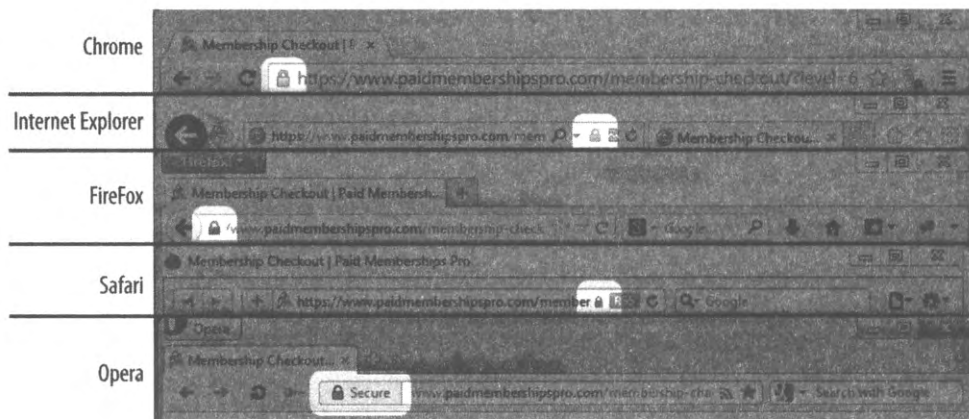


Рис. 8.2. Значок замка в разных браузерах

- ◆ Google и другие поисковые системы начали давать более высокий рейтинг сайтам, которые обслуживаются полностью по протоколу HTTPS.

- ♦ Применение протокола HTTPS уже не ведет к повышению нагрузки на процессор вашего сервера. Стек веб-серверов был обновлен так, что теперь они лучше справляются с протоколом HTTPS и даже рассчитывают на него, так что вы уже не можете указывать на большое время загрузки страниц в качестве предлога для отключения протокола HTTPS на своем сайте.

Один каталог и для HTTPS-, и для HTTP-трафика

Наряду с установкой сертификата от ЦС, для настройки протокола SSL также следует направить свой HTTPS-каталог на свой HTTP-каталог, используя *символическую ссылку*, или *"симвлинк"* (от англ. "symbolic link"). Симвлинк выполняет ту же роль, что и ярлыки в операционной системе Windows — не являясь каталогом, он указывает на некоторый другой каталог.

В конечном итоге создание симлинка для HTTPS-каталога приведет к тому, что при посещении пользователями сайтов `https://yoursite.com` и `http://yoursite.com` будут загружаться одни и те же исходные файлы `*.php`. Ваш сервер обеспечит шифрование трафика при наличии HTTPS-ссылки, а WordPress и тот плагин электронной коммерции, который вы, возможно, используете, обеспечат отображение соответствующей защищенной версии страницы при ее обслуживании по протоколу SSL.

Допустим, что ваш HTTP-каталог называется "html", и вы хотите, чтобы ваш HTTPS-каталог назывался "ssl_html". В таком случае создайте символическую ссылку для этого каталога с помощью следующей команды Linux: `ln -s http ssl_http`.

Авторизация и доступ к панели администратора WordPress по протоколу SSL

Чтобы защитить персональные данные посетителей вашего сайта, вы, как минимум, должны обслуживать по протоколу SSL страницу оплаты. Также можно настроить WordPress на использование протокола SSL на странице авторизации, в панели администратора, в рамках всего сайта и только на отдельных страницах.

Чтобы задействовать протокол SSL для авторизации в WordPress, присвойте константе `FORCE_SSL_LOGIN` значение `true` в файле `wp-config.php`. Поместите следующую строку кода перед комментарием "That's all, stop editing! Happy blogging." в конце этого файла:

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

Чтобы активировать протокол SSL не только на странице авторизации, *но и* в панели администратора, задайте константу `FORCE_SSL_ADMIN`:

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



В данном случае константа `FORCE_SSL_ADMIN` используется вместо константы `FORCE_SSL_LOGIN`. Значение `true` следует присвоить либо первой, либо второй константе. Если константе `FORCE_SSL_LOGIN` будет присвоено значение `false`, а константе `FORCE_SSL_ADMIN` — значение `true`, то страница авторизации все равно будет обслуживаться по протоколу SSL.

Отладка проблем с протоколом HTTPS

Теперь давайте напишем пример небольшой функции, которая будет фильтровать генерируемые системой WordPress URL-адреса так, чтобы они использовали тот же протокол, что и текущая страница. Как уже говорилось ранее, при появлении URL-адресов вида `https://yoursite.com/some-page` (для протокола HTTP) на странице с адресом вида `https://yoursite.com/checkout` (для протокола HTTPS) браузер выводит предупреждение системы безопасности.

Пример

```
function my_https_filter($s) {
    if(is_ssl())
        return str_replace("http:", "https:", $s);
    else
        return str_replace("https:", "http:", $s);
}

add_filter('bloginfo_url', 'my_https_filter');
add_filter('wp_list_pages', 'my_https_filter');
add_filter('option_home', 'my_https_filter');
add_filter('option_siteurl', 'my_https_filter');
add_filter('logout_url', 'my_https_filter');
add_filter('login_url', 'my_https_filter');
add_filter('home_url', 'my_https_filter');
```

Предоставляемая системой WordPress функция `is_ssl()` возвращает значение `true` в случае загрузки текущей страницы по протоколу HTTPS. А если быть точнее, эта функция проверяет, присвоено ли глобальной переменной `$_SERVER['HTTPS']` значение `on` или `1`, а глобальной переменной `$_SERVER['SERVER_PORT']` — значение `443`. Некоторые серверные конфигурации балансировщиков нагрузки и обратных прокси-серверов не производят должную настройку этих глобальных переменных языка PHP при загрузке HTTPS-страниц. Это можно исправить, добавив следующий код в файл `wp-config.php`:

```
if (isset($_SERVER['HTTP_X_FORWARDED_PROTO'])
    && $_SERVER['HTTP_X_FORWARDED_PROTO'] == 'https') {
    $_SERVER['HTTPS'] = 'on';
}
```

Наш HTTPS-фильтр заменяет `"http:"` на `"https:"`, и наоборот, используя для этого функцию `str_replace()`. Мы настроили запуск этого фильтра для ряда встроенных хуков системы WordPress в тех местах кодовой базы WordPress, где генерируются URL-адреса.

Если URL-адреса создаются в кастомном коде вашего приложения, не забудьте проверить корректность генерируемых URL-адресов с помощью функции `home_url()`, и запустить для них фильтр `my_https_filter`.

"Атомный" способ устранения ошибок протокола SSL

Функция `my_https_filter()` обеспечивает выбор надлежащего протокола для отображаемых на странице ссылок. Однако иногда исходные URL-адреса вида `"http://..."` могут оказаться жестко заданными в ваших постах, или может оказаться, что ваш плагин не использует должным образом встроенные функции WordPress при создании URL-адресов в рамках одного сайта или при загрузке JavaScript- или CSS-файлов. В обнаружении таких ошибок может помочь показанная на рис. 8.3 панель инструментов разработчика браузера Chrome.

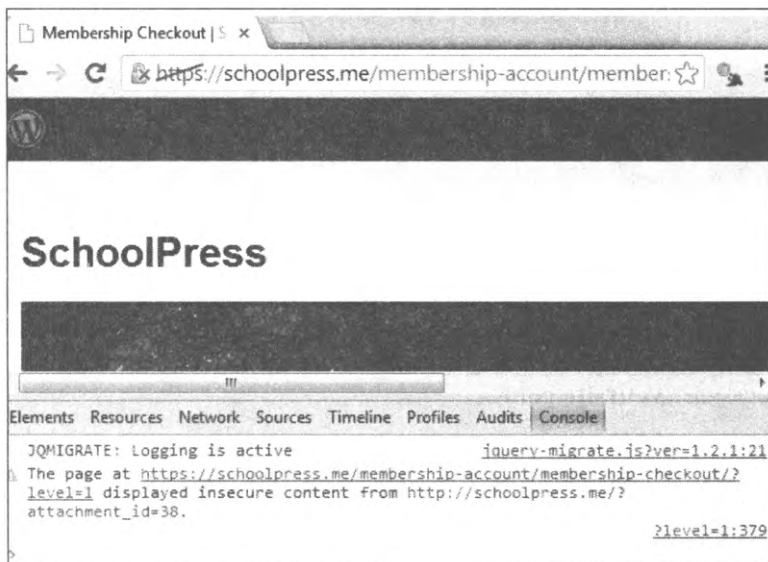


Рис. 8.3. Отображение ошибки SSL на панели инструментов разработчика браузера Chrome

В таких случаях можно попытаться найти и исправить каждый такой "неправильный" URL-адрес в постах или программном коде, или задать относительные URL-адреса или обеспечить надлежащий протокол для выводимых в клиентской части адресов с помощью функции WordPress. Однако иногда проще воспользоваться "атомным" способом, как показано в следующем примере.

Пример

```
constant('MY_SITE_DOMAIN', 'yoursite.com');

function my_NuclearHTTPS() {
    ob_start("my_replaceURLsInBuffer");
}

add_action("init", "my_NuclearHTTPS");

function my_replaceURLsInBuffer($buffer) {
    global $bsecure;
```



```

        // заменяем URL-адреса, только если страница является защищенной
        if(is_ssl())
    {
        /*
        заменяем все ссылки следующего вида:
        * http://yoursite.com
        * http://anysubdomain.yoursite.com
        * http://any.number.of.sub.domains.yoursite.com
        */
        $buffer = preg_replace(
            '/http:\/\/\:\/\/([a-zA-Z0-9\.\-]*'.str_replace('.', '\.', MY_SITE_DOMAIN).')/i',
            'https://$1',
            $buffer
        );
    }
    return $buffer;
}

```

Сначала мы проверяем, определена ли у нас константа `MY_SITE_DOMAIN`, и присваиваем ей домен второго уровня вашего сайта. Функция `site_url()` системы WordPress выдала бы адрес вида **`http://<www.yoursite.com>`**, однако в данном случае нам требуется только сегмент **`yoursite.com`**.

Затем функция `my_NuclearHTTPS()` запускает хук `init` и включает буферизацию вывода с помощью функции `ob_start()` языка PHP. Под буферизацией вывода понимается то, что весь вывод, генерируемый PHP-кодом (например, вызовами функции `echo()` или встроенным HTML-кодом) поступает в буферную строку, а не напрямую в браузер. Затем, после того, как PHP-код завершает генерирование всего вывода (или, если это происходит раньше, при вызове функции `ob_end_flush()`), буферная строка передается в функцию обратного вызова — в данном случае, в функцию `my_replaceURLsInBuffer()`.

Функция `my_replaceURLsInBuffer()` фильтрует буферную строку, заменяя в *каждой* ссылке сегмент `"http:"` на сегмент `"https:"`. Хитрое регулярное выражение в вызове функции `preg_replace()` обеспечивает фильтрацию ссылок на все поддомены в рамках того же домена (именно для этого нам нужно было определить константу `MY_SITE_DOMAIN`).

Наверно, вы уже поняли, почему этот способ называется "атомным". Вместо того, чтобы последовательно искать и исправлять "неправильные" URL-адреса в исходном коде приложения, мы просто исправляем сразу все URL-адреса перед отправкой вывода в браузер. Это приведет к некоторому спаду производительности, величина которого зависит от объема генерируемого вами HTML-вывода. Однако этот метод может оказаться полезным в тех случаях, когда у вас не будет лишнего времени, особенно, если вы используете много сторонних плагинов, и не можете или не хотите исправлять их код, чтобы обеспечить правильное генерирование URL-адресов на своем сайте.



Плагин Really Simple SSL (oreil.ly/dMfNq) предлагает многие из упомянутых ранее способов настройки протокола HTTPS, а также ряд других инструментов, которые помогут вам обеспечить надлежащую работу протокола HTTPS на вашем сайте на базе WordPress.

Резервируйте все!

Необходимо регулярно проводить резервное копирование содержимого вашего сайта (вашей базы данных), а также папки wp-content. Это существенно упростит восстановление вашего сайта в том случае, если он выйдет из строя в результате хакерской атаки. Мы рекомендуем осуществлять плановое резервное копирование не реже одного раза в неделю, однако вы можете делать это реже или чаще в зависимости от объемов добавляемого вами нового контента. Конечно, идеальным вариантом в любом случае будет ежедневное резервное копирование.

Знаете ли вы, где хранятся ваши резервные копии?

Сможете ли вы действительно восстановить свой сайт, используя имеющиеся резервные копии? Через каждые несколько месяцев производите попытку восстановить свой сайт из резервных копий. Это позволит гарантировать, что они останутся в работоспособном состоянии, что вы будете копировать все необходимые данные, и что сможете быстро восстановить свой сайт из резервных копий. Мы слышали немало страшных историй о том, как, рассчитывая на свои резервные копии, владельцы сайтов в итоге обнаруживали, что резервные копии повреждены, включают в себя не все данные, либо по какой-либо иной причине непригодны для восстановления сайта.

Сканируйте, сканируйте и еще раз сканируйте!

Сканирование или отслеживание состояния приложения играет важную роль в выявлении атак. Если ваше приложение подвергнется атаке, об этом лучше узнать сразу, чтобы можно было быстро решить данную проблему.

Меры по защите веб-приложения от вредоносных программ следует принимать заранее. Чтобы избавить себя от лишнего вмешательства в этот процесс, воспользуйтесь одним из сервисов, способных сканировать веб-приложения в автоматическом режиме. Мы рекомендуем сервис Sucuri (sucuri.net). Он способен не только выявить вредоносное ПО и уведомить вас в случае заражения, но и автоматически удалить его. Исполнительный директор компании Sucuri Тони Перес также может похвастаться опытом службы в американской морской пехоте и хорошим владением боевыми искусствами, так почему бы не доверить свою защиту этому сервису? Компания Sucuri также предлагает отличный защитный плагин для WordPress (bwwp.com/sucuri-plugin).

Полезные плагины для обеспечения безопасности

В следующих разделах мы рассмотрим ряд плагинов для WordPress, которые являются наиболее полезными и мощными в плане повышения безопасности приложения, а также его быстрого восстановления после вредоносных атак.

Плагины для блокировки спама

Спам — проблема для любого сайта в Интернете. Справиться с ней помогут следующие плагины.

Akismet

Этот плагин позволяет предотвратить появление на вашем сайте спам-комментариев. Поскольку его разработала компания Automattic, в свое время создавшая саму систему WordPress, он входит в стандартный комплект поставки этой CMS. Плагин Akismet устанавливается на вашем сайте, однако вам нужно будет активировать его, пройдя регистрацию и получив API-ключ на сайте akismet.com. API-ключ предоставляется бесплатно для некоммерческих сайтов, и за небольшую плату в случае коммерческих сайтов. Akismet действует следующим образом: каждый размещаемый на сайте комментарий прогоняется через серию тестов, подтверждающих, что это действительно комментарий; весь выявленный спам автоматически перемещается в папку для спама в вашей панели администратора. Это позволит вам не тратить массу времени на проверку каждого комментария в отношении того, является ли он спамом, или нет.

Bad Behavior

Плагин Bad Behavior ([bwawwp.com/bad-behavior](https://b2eville.com/bad-behavior/)) не допускает появление на сайте ссылочного спама и работает наиболее эффективно в сочетании с каким-либо дополнительным сервисом блокировки спама. Данный плагин выявляет не только характерное для спама содержимое, но и методы рассылки спама, а также используемое для этой цели ПО; все перечисленное блокируется.

Плагины для резервного копирования

Резервные копии могут оказаться очень полезными в случае взлома вашего сайта. Вот пара популярных плагинов для резервного копирования.

BackupBuddy

Как уже упоминалось в *главе 3*, BackupBuddy ([bwawwp.com/backup-b](https://backupbuddy.com/)) — это премиум-плагин, позволяющий вам создавать резервные копии всего содержимого вашего сайта на базе WordPress с целью безопасного сохранения, восстановления или

перемещения вашего сайта. Резервное копирование может выполняться по расписанию, а создаваемые при этом файлы могут отправляться на ваш адрес электронной почты, в облачное хранилище наподобие Dropbox или на FTP-сервер. Предлагаемая этим плагином функция восстановления позволяет легко восстанавливать темы, виджеты и плагины. BackupBuddy дает возможность вам перенести сайт на новый сервер или домен из панели администратора WordPress, что весьма удобно, если вы разрабатываете сайты на сервере разработки и переносите их в эксплуатационное окружение при введении их в действие.

VaultPress

VaultPress (vaultpress.com) — это еще один плагин от компании Automattic, дающий пользователю возможность производить резервное копирование всего содержимого своего сайта в режиме реального времени с сохранением копий на облачных серверах. После установки, он будет автоматически выявлять любые изменения в содержании и настройках сайта, и отражать эти изменения в резервной копии. Данный плагин также позволяет восстановить базу данных одним щелчком мыши в случае взлома вашего сайта. Поскольку это премиум-плагин, за его использование взимается некоторая плата, причем предлагается несколько вариантов оплаты. Расширенная версия плагина также позволяет ежедневно сканировать сайта на наличие проблем безопасности и устранять выявленные проблемы.

Плагины-брандмауэры/сканеры

Перечисленные далее плагины служат для выявления и предотвращения тех разновидностей автоматизированных атак, которым подвергается каждый веб-сайт в Интернете.

WordFence

Плагин WordFence (oreil.ly/4vXC9) выступает в роли брандмауэра для вашего сайта, сканируя входящий трафик и блокируя любые виды вредоносных запросов. Вы также можете выполнить сканирование своего сайта, выявляющее все его слабые места в плане безопасности. Выбрав платную версию плагина, вам будет доступны несколько дополнительных премиум-инструментов и более часто обновляемая база данных вредоносных программ и уязвимостей.

All In One WP Security & Firewall

Плагин All In One WP Security & Firewall (oreil.ly/Yc7JP) предлагает функции брандмауэра и сканера, сходные с таковыми функциями плагина WordFence. Также предлагаются инструменты для усиления защиты авторизации и пользователей. Данный плагин также помогает в решении такой важной проблемы безопасности, как изменение префикса таблиц базы данных, что может вызывать у вас затруднения, если вы не знакомы со стандартной структурой базы данных.

Exploit Scanner

Плагин Exploit Scanner компании Automatic (bwawwp.com/exploit-scan) сканирует все файлы вашего сайта и уведомляет вас о любых обнаруженных потенциальных угрозах.

Плагины для защиты авторизации и пароля

Представленные далее плагины специально предназначены для ограничения доступа к странице авторизации и всей панели администратора WordPress.

Limit Login Attempts

Плагин Limit Login Attempts (bwawwp.com/limit-login) хорошо подходит для защиты от атак методом прямого перебора, например, от автоматизированных скриптов, осуществляющих попытки войти в WordPress, используя случайные сочетания слов. По умолчанию WordPress не ограничивает количество попыток авторизации, и данный плагин позволяет ограничить количество таких попыток. Если определенный пользователь совершит x неудачных попыток авторизации, то возможность авторизации будет заблокирована для него на некоторый заданный промежуток времени.

AskApache Password Protect

В отличие от других защитных плагинов для WordPress, плагин AskApache Password Protect (bwawwp.com/ask-apache) обеспечивает защиту от атак на сетевом уровне, а не на уровне сайта. Вы выбираете уникальное имя пользователя и пароль, после чего они используются для защиты доступа к странице авторизации и всей папке wp-admin. Для работы данного плагина требуется веб-сервер Apache и хостинг с поддержкой файлов *.htaccess.

Написание безопасного кода

Важно, чтобы любой кастомный программный код был безопасным и не давал хакерам возможностей для взлома. Если вы со всей серьезностью отнесетесь к изложенным далее рекомендациям, то ваш код будет достаточно хорошо защищен от возможных атак.

Проверяйте полномочия пользователей

Каждый из ваших пользователей обладает уникальным набором стандартных или кастомных ролей и полномочий. Если вы пишете код, предоставляющий кастомную функциональность администраторам вашего приложения, убедитесь, что эти возможности предоставляются исключительно администраторам. Несколько встроенных функций WordPress позволяют выяснить, назначена ли пользователю определенная роль или полномочие. Все эти функции определены в файле wp-

includes/capabilities.php и возвращают логическое значение, указывающее, обладает ли пользователь ролью или полномочием с переданным в качестве параметра названием. Этим функциям можно передавать названия любых дефолтных или кастомных ролей и полномочий.

- ◆ `user_can($user, $capability)` — проверяет, назначена ли определенная роль или полномочие определенному пользователю.
- ◆ `$user` — обязательный целочисленный параметр; идентификатор или объект пользователя.
- ◆ `$capability` — обязательный строковый параметр; название полномочия или роли.
- ◆ `current_user_can($capability)` — проверяет, назначена ли определенная роль или полномочие текущему пользователю.
- ◆ `$capability` — обязательный строковый параметр; название полномочия или роли.
- ◆ `current_user_can_for_blog($blog_id, $capability)` — проверяет, назначена ли определенная роль или полномочие текущему пользователю на определенном сайте многосайтовой сети.
- ◆ `$user` — обязательный целочисленный параметр; идентификатор блога.
- ◆ `$capability` — обязательный строковый параметр; название полномочия или роли.

В представленном далее примере мы хотим запретить доступ к серверной части приложения для обычных пользователей. Они должны использовать только кастомный пользовательский интерфейс, созданный нами в рамках темы клиентской части приложения. Поэтому в случае, когда не обладающий ролью администратора пользователь попытается зайти в папку /wp-admin, мы перенаправляем его назад в клиентскую часть.

Пример

```
function schoolpress_admin_check() {  
    global $user_ID;  
    if (! user_can($user_ID, 'administrator')) {  
        wp_redirect(site_url());  
    }  
}  
  
add_action('admin_init', 'schoolpress_admin_check');
```



Вместо того чтобы проверять наличие роли администратора, многие плагины для WordPress проверяют, назначено ли пользователю полномочие **manage_options**. Это дает тот же результат, поскольку WordPress все равно по умолчанию назначает это полномочие только администратору. В то же время проверка на наличие полномочия **manage_options** вместо проверки на наличие роли **administrator** гарантирует работу кода на сайтах с кастомными ролями.

Более подробные сведения о стандартных ролях и полномочиях, предоставляемых в WordPress по умолчанию, можно найти в *главе 6* и на соответствующей странице Кодекса WordPress (bawwp.com/roles-caps).

Кастомные инструкции SQL

В некоторых случаях возможности встроенных функций WordPress для работы с базой данных не смогут покрыть все ваши потребности. В зависимости от того, что именно вы создаете, в таком случае часто необходимы кастомные инструкции SQL. При написании собственных инструкций SQL важно проследить за тем, чтобы была исключена любая возможность для внедрения SQL-кода. Прежде всего, всегда используйте объект `$wpdb` и не забывайте экранировать и подготавливать все кастомные инструкции SQL.

Как упоминалось в *главе 3*, объект `$wpdb` может использоваться для доступа к любым стандартным или кастомным таблицам базы данных WordPress и предлагает для этой цели простые методы. Если вашим кастомным запросам будут передаваться какие-либо динамические значения, важно проследить за тем, чтобы эти динамические значения подвергались санации и экранированию с помощью функции `esc_sql()` или метода `prepare()`. Санация и экранирование динамических значений исключит вероятность того, что эти значения будут содержать недопустимые символы или вредоносный SQL-код, способный перехватить ваш запрос (внедренный SQL-код).

Функции `esc_sql()` и `$wpdb->prepare()` были подробно рассмотрены в *главе 3*.

Валидация, санация и экранирование данных

Не доверяйте своим пользователям! Повторим еще раз: *никогда не доверяйте своим пользователям*, если не хотите, чтобы ваше веб-приложение, веб-сайт или блог стал "переносчиком" вредоносного ПО.

Подвергайте валидации, санации и экранированию буквально каждый фрагмент данных, записываемых вами в базу данных и извлекаемых из нее. Вы должны следить за тем, чтобы данные, отправляемые пользователями в базу данных, предоставлялись в требуемом формате. Базу данных "не волнует", какую именно информацию вы сохраняете, если эти данные относятся к заданному типу.

Например, допустим, что у вас есть кастомная форма для получения информации о пользователе, содержащая текстовое поле для ввода даты рождения. Вы планируете сохранять дату рождения как пользовательские метаданные в столбце `meta_value` таблицы `wp_usermeta`. Столбцу `meta_value` присвоен тип данных `longtext`, что позволяет сохранять в нем сверхдлинные¹ значения, и при этом безразлично, что именно вы сохраняете. Это вы, как разработчик, должны позаботиться о том, чтобы данные, сохраняемые как дата рождения, представляли собой дату, а не что-либо иное.

Однако что именно имеется в виду под валидацией, санацией и экранированием?

Валидация — процесс проверки данных, получаемых от конечного пользователя, на соответствие тому формату, в котором они должны предоставляться. Данные должны подвергаться валидации перед сохранением их в базе данных.

¹ Если быть точным, то размер этих "сверхдлинных" значений может достигать до 4 ГБ.

Санация — процесс очистки данных, получаемых от конечного пользователя, перед сохранением их в базе данных или использованием их в приложении.

Экранирование — процесс очистки уже имеющихся у вас данных перед отображением их конечному пользователю, сохранением их в базе данных или передачей их некоторому API.

Валидации и санации необходимо подвергать все данные, которые поступают в ваше приложение через отправляемые формы, параметры URL-адресов или вызовы API. Экранированию необходимо подвергать все данные, которые вы помещаете в базу данных или выводите на экран. При извлечении данных из базы данных их необходимо подвергать санации на тот случай, если вы случайно сохраните не прошедшие санацию данные.

Хотя в языке PHP имеются функции для валидации и санации, WordPress предлагает для этой цели собственные встроенные вспомогательные функции. Поскольку эта книга посвящена системе WordPress, давайте рассмотрим некоторые из этих функций.

Большинство вспомогательных функций для санации и экранирования определено в файле `wp-includes/formatting`.

- ◆ `esc_url($url, $protocols = null, $context = 'display')` — проверяет и очищает URL-адрес. Проверяет его на соответствие требуемому протоколу, удаляет недопустимые символы и кодирует специальные символы. Используйте эту функцию при отображении URL-адреса конечному пользователю.
 - `$url` — обязательный строковый параметр; URL-адрес, который нужно подвергнуть очистке;
 - `$protocols` — опциональный массив имен допустимых протоколов. По умолчанию допустимы следующие протоколы: `http`, `https`, `ftp`, `ftps`, `mailto`, `news`, `irc`, `gopher`, `nntp`, `feed`, `telnet`, `mms`, `rtsp`, `svn`;
 - `$context` — опциональный строковый параметр, указывающий, как будет использоваться URL-адрес. Значение по умолчанию — `display`; при этом для URL-адреса запускается функция `wp_kses_normalize_entities()` и символ `&` заменяется на код `&`, а символ `'` — на код `'`.
- ◆ `esc_url_raw($url, $protocols = null)` — данная функция вызывает функцию `esc_url()`, передавая объект `db` в качестве параметра `$context`. Не используйте эту функцию для отображения URL-адресов конечному пользователю; вызывайте ее только в запросах к базе данных.
- ◆ `esc_html($text)` — экранирует блоки HTML-кода в любом контенте. Данная функция представляет собой короткую и аккуратную обертку для функции `_wp_specialchars()`, которая преобразует коды специальных символов в соответствующие HTML-сущности.
 - `$text` — обязательный строковый параметр; текст, в котором нужно экранировать HTML-теги.

- ◆ `esc_js($text)` — экранирует строки встроенного JavaScript-кода. Экранируемая строка должна быть заключена в одинарные кавычки.
 - `$text` — обязательный строковый параметр; текст, в котором нужно экранировать одинарные кавычки, специальные символы HTML (" < > &), и исправить окончания строк.
- ◆ `esc_attr($text)` — экранирует HTML-атрибуты и кодирует такие символы, как <, >, &, " и '. Данная функция необходима при включении значений в элементы форм для ввода таких данных, как идентификаторы, имена, заголовки, Alt-коды и числовые значения.
 - `$text` — обязательный строковый параметр; текст, в котором нужно экранировать HTML-атрибуты.
- ◆ `esc_textarea($text)` — экранирование для значений `textarea`. Кодирует текст для использования внутри элемента `<textarea>`.
 - `$text` — обязательный строковый параметр; текст, в котором нужно экранировать HTML-код.
- ◆ `sanitize_option($option, $value)` — данная функция может использоваться для санации значения любой заранее определенной опции WordPress. Значение санируется с помощью различных функций, в зависимости от типа обрабатываемой опции.
 - `$option` — обязательный строковый параметр; имя опции;
 - `$value` — обязательный строковый параметр; подвергаемое санации значение опции.
- ◆ `sanitize_text_field($str)` — санирует любую строку, введенную пользователем или извлеченную из базы данных. Проверяет на ошибки в использовании кодировки UTF-8; преобразует одиночные символы < в соответствующие HTML-сущности; удаляет все теги; удаляет разрывы строк, символы табуляции и лишние пробелы; удаляет октеты.
 - `$str` — обязательный строковый параметр; строка, которую нужно подвергнуть санации.
- ◆ `sanitize_user($username, $strict = false)` — данная функция очищает от недопустимых символов имя пользователя.
 - `$username` — обязательный строковый параметр; имя пользователя, которое нужно подвергнуть санации;
 - `$strict` — опциональный логический параметр; при присвоении данному параметру значения `true` в имени пользователя допускается использование только определенного набора символов.
- ◆ `sanitize_title($title, $fallback_title = '')` — санирует заголовок, удаляя из него HTML- или PHP-теги, либо возвращает запасной вариант заголовка для переданной строки:
 - `$title` — обязательный строковый параметр; заголовок, который нужно подвергнуть санации;

- `$fallback_title` — опциональный строковый параметр; строка, используемая в том случае, если заголовок окажется пустым.
- ◆ `sanitize_email($email)` — saniрует адрес электронной почты, удаляя из него символы, которые являются недопустимыми для адресов электронной почты.
 - `$email` — адрес электронной почты, который нужно подвергнуть санации.
- ◆ `sanitize_file_name($filename)` — saniрует имя файла, заменяя пробелы на тире. Удаляет специальные символы, которые являются недопустимыми для имен файлов в некоторых операционных системах, и специальные символы, требующие специального экранирования при использовании в командной строке. Заменяет пробелы и последовательности из нескольких тире на одно тире. Удаляет лишние точки, тире и подчеркивания в начале и конце имени файла.
 - `$filename` — обязательный строковый параметр; имя файла, которое нужно подвергнуть санации.
- ◆ `wpkses($string, $allowed_html, $allowed_protocols = array ())` — оставляет в переданной строке только допустимые имена HTML-элементов, имена и значения атрибутов, а также только полноценные HTML-сущности. Перед вызовом этой функции необходимо удалить все символы косой черты из "волшебных" кавычек PHP.
 - `$string` — обязательный строковый параметр; строка, которую нужно отфильтровать с помощью `kses`;
 - `$allowed_html` — обязательный массив допустимых HTML-элементов;
 - `$allowed_protocols` — опциональный массив имен протоколов, допустимых для URL-адресов в фильтруемой строке. По умолчанию допустимы следующие протоколы: `http`, `https`, `ftp`, `mailto`, `news`, `irc`, `gopher`, `nnntp`, `feed`, `telnet`, `mms`, `rtsp`, `svn`. Данный список включает в себя все распространенные протоколы ссылок, за исключением протокола `javascript`, использование которого следует запретить для непроверенных пользователей.
- ◆ `wpkses_post($data)` — функция saniрует переданные ей данные, оставляя в них те HTML-теги и протоколы, которые допустимы для раздела контента поста на странице редактирования поста. Если вы хотите назначить более строгие правила по сравнению с правилами, заданными для авторов, редакторов и администраторов на вашем сайте, используйте функцию `wpkses()`, передавая ей конкретные теги и протоколы. Если обрабатываемое поле предназначено для администраторов, и вам нужно просто saniровать его так же, как saniруется содержимое поста, примените функцию `wpkses_post()` как более удобный и сокращенный способ.
 - `$data` — обязательный строковый параметр; строка, которую нужно отфильтровать с помощью `kses`.

В следующем примере приведен код, который подвергает валидации и санации адрес электронной почты.

```
// имитируем добавление пользователем адреса электронной
// почты "jason @ stranger$studios.com"
$user_email = 'jason @ stranger$studios.com';

// проверяем, является ли этот адрес допустимым адресом электронной почты
$valid_email = is_email($user_email);

// выясняем, что это не так, поскольку функция is_email()
// возвращает отрицательный результат
if (! $valid_email)
    echo 'invalid email<br />';

// пробуем еще раз, на этот раз с санацией адреса электронной почты
$user_email = 'jason @ stranger$studios.com';

// пытаемся исправить недопустимый адрес электронной почты
// с помощью функции sanitize_email()
$user_email = sanitize_email($user_email);
$valid_email = is_email($user_email);
if (! $valid_email)
    echo 'invalid email<br />';
else
    echo 'valid email: ' . $user_email;
```

Обратите внимание, что в данном примере функция `sanitize_email()` удаляет из недопустимого адреса электронной почты и пробелы, и знаки доллара. Хотя формально возвращаемый адрес электронной почты является допустимым, это все же не реальный адрес электронной почты Джейсона, поскольку данная функция не знает, что знак доллара следует заменить буквой "s". Также обратите внимание, что возвращаемое значение не всегда будет допустимым адресом электронной почты. При отсутствии знака `@` или текста перед этим знаком, а также при отсутствии доменного имени после знака `@` функция возвратит недопустимый адрес электронной почты.

Дополнительную информацию о валидации, санации и экранировании данных можно найти в Кодексе WordPress (oreil.ly/hGyO9).

Одноразовые коды

Одноразовые коды (nonce, number used once) играют важную роль в защите приложения от атак с подделкой межсайтовых запросов (cross-site request forgery, CSRF). Обычно ваши серверные скрипты для обработки формы обрабатывают формы вашего собственного сайта. Пользователи заходят на ваш сайт, авторизуются и отправляют форму для выполнения определенного действия на вашем сайте. Однако

если ваш серверный код будет просто искать значения `$ _POST`, чтобы определить, какое действие нужно совершить, то эти значения можно будет предоставить с помощью *любой* формы, даже формы другого сайта.

Первую линию защиты при этом составляет проверка в отношении того, авторизовался ли пользователь, и обладает ли он полномочиями на выполнение запрашиваемого действия. Однако эта проверка может оказаться недостаточной для защиты от CSRF-атак, поскольку вы можете уже пройти авторизацию своего сайта на базе WordPress (например, на фоновой вкладке), когда некоторый вредоносный код отправит с другого сайта/вкладки запрос формы с правильной переменной `$ _POST` с тем, чтобы разослать спам вашим друзьям, удалить аккаунт или выполнить какие-либо иные нежелательные действия.

Нам нужно каким-то образом сделать так, чтобы запрос мог поступать от нашего сайта на базе WordPress, но не какого-либо иного сайта. Именно для этой цели предназначены одноразовые коды. Общая схема применения одноразового кода выглядит следующим образом:

- ◆ При каждой загрузке страницы генерируется одноразовый код в виде строки.
- ◆ Эта строка одноразового кода добавляется в форму как скрытый элемент.
- ◆ При обработке присланной формы точно так же генерируется одноразовый код, который проверяется на совпадение с кодом, извлеченным из формы.

Поскольку одноразовый код генерируется на основе комбинации секретных соляных ключей из файла `wp-config.php` и информации о времени сервера, злоумышленникам будет очень сложно подобрать строку одноразового кода для своих подложных форм.

Одноразовые коды также пригодны для бесформенных ссылок и Ajax-вызовов. Это делается практически аналогично:

- ◆ При каждой загрузке страницы генерируется одноразовый код в виде строки.
- ◆ Эта строка одноразового кода добавляется в URL-адрес как параметр.
- ◆ При обработке запроса точно так же генерируется одноразовый код, который проверяется на совпадение с кодом, извлеченным из URL-адреса.

WordPress предлагает несколько вспомогательных функций, которые существенно упростят процесс реализации такой защиты независимо от того, что именно вам нужно защитить — формы, ссылки или Ajax-запросы.

- ◆ `wp_create_nonce($action = -1)` — данная функция создает случайный токен, который может быть использован только один раз и размещается в файле `wp-includes/pluggable.php`.
 - `$action` — опциональный строковый или целочисленный параметр, указывающий, какое действие должно выполняться для создаваемого одноразового кода. Для повышения безопасности рекомендуется всегда указывать выполняемое действие.

Пример

```
function schoolpress_footer_create_nonce(){
    $nonce = wp_create_nonce('random_nonce_action');
    $url = add_query_arg(array('sp_nonce' => $nonce));
    echo '<p><a href="' . $url . '">Verify this Nonce</a></p>';
}
add_action('wp_footer', 'schoolpress_footer_create_nonce');
```

◆ `wp_verify_nonce($nonce, $action = -1)` — данная функция проверяет, был ли использован правильный одноразовый код в пределах выделенного промежутка времени. Если переданный одноразовый код является правильным, и проверка дает положительный результат, функция возвращает значение, приравненное к значению `true`². В противном случае она возвращает значение `false`. Данная функция определена в файле `wp-includes/pluggable.php`.

- `$nonce` — обязательный строковый параметр; проверяемый одноразовый код;
- `$action` — опциональный строковый или целочисленный параметр, описывающий выполняемое действие; должен совпадать с действием, указанным при создании одноразового кода.

Пример

```
function schoolpress_init_verify_nonce(){
    if (isset($_GET['sp_nonce']))
        && wp_verify_nonce($_GET['sp_nonce'], 'random_nonce_action')) {
        echo 'You have a valid nonce!';
    } else {
        echo 'You have an invalid nonce!';
    }
}
add_action('init', 'schoolpress_init_verify_nonce');
```

◆ `check_admin_referer($action = -1, $query_arg = '_wpnonce')` — данная функция вызывает функцию `wp_verify_nonce()` для проверки одноразовых кодов, но, помимо этого, также проверяет, находится ли в пределах того же сайта *источник ссылки*, или, иначе говоря, страница, с которой был осуществлен переход на текущую страницу. Данная функция определена в файле `wp-includes/pluggable.php`.

² Функция `wp_verify_nonce()` возвращает значение 1, если возраст одноразового кода не превышает 12 часов. Если возраст одноразового кода составляет от 12 до 24 часов, возвращается значение 2. Если возраст одноразового кода превышает 24 часа, возвращается значение `false`. Это позволяет вам проверить результат на равенство значению `true`, либо проверить его на равенство значению 1, если нужно, чтобы одноразовые коды были более свежими.

- `$action` — опциональный строковый параметр, однако вам потребуется указать действие одноразового кода для выполнения проверки;
- `$query_arg` — опциональный строковый параметр; аргумент запроса, содержащий одноразовый код.

Пример

```
// Проверка созданного ранее одноразового кода "sp_nonce"
function schoolpress_init_check_admin_referer(){
    if (isset($_GET['sp_nonce']) &&
        check_admin_referer('random_nonce_action', 'sp_nonce')) {
        echo '<p>You have a valid nonce!</p>';
    } else {
        echo '<p>You have an invalid nonce!</p>';
    }
}

add_action('init', 'schoolpress_init_check_admin_referer');
```

◆ `wp_nonce_url($actionurl, $action = -1)` — данная функция вызывает функцию `wp_create_nonce()` и добавляет созданный одноразовый код в любой URL-адрес. Если вы создаете какие-либо действия на основе строки запроса, при этом всегда следует привязывать к URL-адресу одноразовый код с помощью данной функции.

- `$actionurl` — обязательный строковый параметр; URL-адрес, к которому нужно добавить действие одноразового кода;
- `$action` — опциональный строковый параметр; название действия. Рекомендуется всегда указывать действие.

Данная функция определена в файле `wp-includes/functions.php`.

Пример

```
// Пример простого URL-адреса со строкой запроса
function schoolpress_footer_nonce_url(){
    $url = wp_nonce_url(
        add_query_arg(array('action' => 'get_users')),
        'get_users_nonce'
    );
    echo '<p><a href="' . esc_url($url) . '">Get Users</a></p>';
}

add_action('wp_footer', 'schoolpress_footer_nonce_url');
```

// действие строки запроса

```
function schoolpress_footer_nonce_url_action(){
    // проверяем, является ли действие строки запроса
    // действием get_users, и проверяем одноразовый код
```

```

    if (isset($_GET['action']))
        && 'get_users' == $_GET['action']
        && check_admin_referer('get_users_nonce')) {
        echo 'Your action: ' . esc_html($_GET['action']);
        // или получаем пользователей и отображаем их здесь...
    }
}

add_action('init', 'schoolpress_footer_nonce_url_action');

```

Обратите внимание, что в данном примере мы с помощью функции `esc_html()` выводим сведения о действии, переданном в строку запроса. Обычно мы не используем в примерах кода функции для экранирования, поскольку они затрудняют как чтение кода, так и понимание того, что делает код. Однако здесь мы отошли от этого правила, поскольку данная глава посвящена безопасности, и небрежный вывод параметров URL-адреса — один из основных способов сделать сайт уязвимым для межсайтового скриптинга.

Также обратите внимание, что мы вызываем функцию `esc_url()`, когда выводим в ссылку URL-адрес с одноразовым кодом. Даже если для создания URL-адреса используются функции, предоставленные системой WordPress, необходимо экранировать URL-адрес перед его выводом. Функция `esc_url()` не позволяет вызывать ее многократно, что могло бы испортить URL-адрес.

♦ `wp_nonce_field($action = -1, $name = '_wpnonce', $referer = true, $echo = true)` — данная функция извлекает или отображает содержимое скрытого поля формы с одноразовым кодом. Поскольку она использует функцию `wp_create_nonce()`, при работе с формами эта удобная вспомогательная функция всегда пригодится.

Поле одноразового кода служит для подтверждения того, что источник содержимого формы находится на текущем сайте, а не где-то за его пределами. Одноразовый код не гарантирует абсолютно надежной защиты, но может обеспечить защиту в большинстве случаев. В силу этого мы настоятельно рекомендуем поле одноразового кода в формах.

Параметры `$action` и `$name` являются опциональными, но для повышения безопасности настоятельно рекомендуется всегда их задавать. Конечно, проще вызвать данную функцию без параметров, поскольку они не требуются для проверки одноразового кода, но если злоумышленники узнают, что представляет собой дефолтная реализация, они легко смогут обойти ваш одноразовый код и причинить ущерб.

В качестве имени поля используется любое предоставленное вами значение параметра `$name`. Содержимое поля представляет собой значение, присвоенное при создании одноразового кода. Данная функция определена в файле `wp-includes/functions.php`.

- `$action` — опциональный строковый параметр; название действия. Рекомендуется всегда задавать это название;

- `$name` — опциональный строковый параметр; имя одноразового кода. Рекомендуется всегда задавать это имя;
- `$referer` — опциональный логический параметр; указывает, нужно ли задавать для проверки поле источника ссылки. Значение по умолчанию — `true`;
- `$echo` — опциональный логический параметр; указывает, нужно ли отображать или возвращать скрытое поле формы. Значение по умолчанию — `true`.

Пример

```
<?php
// простой пример отправки формы
function schoolpress_footer_form(){
    ?>
    <form method="post">
        <?php // создаем наш одноразовый код
        wp_nonce_field('email_list_form', 'email_list_form_nonce');
        ?>
        <h3>Join our email list</h3>
        Email Address: <input type="text" name="email_address">
        <input type="submit" name="submit_email" value="Submit" />
    </form>
    <?php
}

add_action('wp_footer', 'schoolpress_footer_form');

// действие формы
function schoolpress_footer_form_action(){
    if (isset($_POST['submit_email'])
        && isset($_POST['email_address'])
        && check_admin_referer('email_list_form',
            'email_list_form_nonce')) {
        echo 'You submitted: ' . esc_html($_POST['email_address']);
        // или выполняем здесь обработку формы...
    }
}

add_action('init', 'schoolpress_footer_form_action');
?>
```

◆ `check_ajax_referer($action = -1, $query_arg = false, $die = true)` — одноразовые коды рекомендуются в Ajax-запросах. Данная функция позволяет проверить одноразовый код и источник ссылки при обработке Ajax-запроса. Она определена в файле `wp-includes/pluggable.php`.

- `$action` — опциональный строковый параметр; действие, привязанное к одноразовому коду;

- `$query_arg` — опциональный строковый параметр; указывает, по какому ключу следует искать одноразовый код в массиве `$_REQUEST`;
- `$die` — опциональный логический параметр; указывает, следует ли обрывать обработку Ajax-скрипта в случае, если проверка одноразового кода даст отрицательный результат.

Как вы, возможно, заметили, в приведенных в этой книге примерах часто отсутствуют одноразовые коды и санация данных. Мы поступили так с целью сделать примеры кода короткими и понятными, однако, имейте в виду, что вы *всегда* должны использовать одноразовые коды и санацию данных. Каждый раз, когда вы отправляете кастомную форму или задаете URL-адрес с кастомной строкой запроса, применяйте одноразовые коды, и каждый раз, когда вы записываете выражение вида `$_POST['anything']` или `$_GET['anything']`, помещайте его в функцию санации или экранирования.

JavaScript-фреймворки и рабочий процесс

Сегодня код на языке JavaScript является одним из основных компонентов веб-приложений. С появлением программной платформы *Node.js* для запуска JavaScript на сервере и таких мощных фреймворков, как React, JavaScript-код быстро стал неотъемлемой частью любого современного веб-приложения.

В 2012 году соотношение между PHP- и JavaScript-кодом в версии 3.6 системы WordPress выглядело примерно так: 6/7 PHP-кода и 1/7 JavaScript-кода. В 2018 году новый блочный редактор Gutenberg версии WordPress 5.0 демонстрировал уже совершенно иную картину: примерно 1/9 PHP-кода и 8/9 JavaScript-кода. Хотя очевидно, что данный блочный редактор является "надстройкой" над серверным PHP-кодом, очень показателен тот факт, что крупные новые функции системы WordPress теперь создаются главным образом на языке JavaScript. Можно ожидать, что по мере интеграции блочного редактора с другими компонентами панели администратора WordPress доля JavaScript-кода в ядре системы WordPress возрастет еще больше.

Чем же вызван этот переход на использование JavaScript-кода? Если говорить о клиентской части приложений, то отображение веб-сайта с помощью JavaScript-кода, как правило, требует гораздо меньше ресурсов, чем применение для этой цели PHP-кода. Так, загрузка всей модели HTML DOM при навигации по типичному сайту — довольно расточительный подход, поскольку при этом практически не меняются шапка, подвал, меню и другие элементы страницы. С другой стороны, чтобы создать новую страницу с помощью JavaScript-кода, достаточно лишь загрузить изменившуюся часть страницы и поменять класс, предназначенный для элементов меню. Это делает пользовательский опыт более похожим на работу с обычным приложением, и хорошо подходит для веб-приложений в мобильных сетях с низкой пропускной способностью.



Подход, подразумевающий частичное обновление страниц с помощью Ajax-запросов вместо загрузки новых страниц, иногда называют созданием одностраничного приложения.

Прежде чем мы с головой погрузимся в мир JavaScript-кода, вероятно, стоит упомянуть о том, что написание материала данной главы и поддержание его в актуальном состоянии далось нам ценой больших усилий. Язык программирования и платформа, за которыми закрепилось название "JavaScript", представляют собой сложную совокупность стандартов, фреймворков и инструментов.

С технической точки зрения, каждый отдельный браузер использует собственную версию языка JavaScript с индивидуальным набором поддерживаемых возможно-

стей. Разработчики браузеров, вместе с другими заинтересованными сторонами, развивают стандарт языка JavaScript в рамках Технического комитета 39 (TC39) ассоциации Ecma International. В действительности этот стандарт называется "ECMAScript", и язык JavaScript — лишь одна из его реализаций¹. Последняя версия стандарта ECMAScript на данный момент — ECMAScript2018 (иначе называемый "ES2018" или "ES9"), однако "современным" языком JavaScript принято считать стандарт ES6. Эта версия стандарта ECMAScript была выпущена в 2015 году, и впервые включала в себя много новых возможностей, вызвавших массу оптимизма у JavaScript-разработчиков.

Надеемся, что вы не запутались. Как бы нам ни хотелось обойтись без всех этих лишних подробностей и просто рассказать о JavaScript-коде предельно простым языком, иногда вам все же придется сталкиваться с этими названиями при создании своих приложений, и потому мы должны дать вам хотя бы минимальное представление о том, что они подразумевают.

В данной главе мы дадим определение некоторым наиболее распространенным терминам и инструментам, с которыми вы столкнетесь в ходе написания JavaScript-кода для WordPress. Затем мы чуть подробнее рассмотрим ряд конкретных методов, используемых нами и другими WordPress-разработчиками в течение последних нескольких лет. Наконец, мы рассмотрим некоторые современные фреймворки и схемы работы, получившие широкое распространение в сообществе WordPress-разработчиков.



Под "JavaScript-кодом" в данной главе и в книге в целом понимается любой код, написанный на любой версии языка JavaScript, поддерживаемой клиентским браузером, в том числе код, использующий jQuery, и Ajax-вызовы с использованием jQuery.

Что такое ECMAScript?

ECMAScript — стандарт, опубликованный комитетом TC39 ассоциации Ecma International. На сайте комитета TC39 (oreil.ly/0ZekS) его область деятельности определена следующим образом: "Стандартизация универсального, кросс-платформенного, независимого от производителя языка программирования ECMAScript. Это включает в себя синтаксис языка, семантику, библиотеки и дополнительные вспомогательные технологии".

Наиболее популярная реализация стандарта ECMAScript — язык JavaScript. Фактически, можно считать, что "ECMAScript" — это название, употребляемое для стандартов, а "JavaScript" — название языка программирования, реализуемого веб-браузерами².

¹ Реализациями стандарта ECMAScript также являются языки JScript и ActionScript.

² Мы не знаем, почему прописные буквы "ECMA" присутствуют в названии стандарта ECMAScript и отсутствуют в названии ассоциации Ecma International, но именно в таком виде эти названия приводятся на официальных сайтах.

Что такое ES6?

ES6 — это версия стандарта ECMAScript, опубликованная в 2015 году. Почему мы продолжаем говорить об этой версии стандарта сегодня? Дело в том, что стандарт ES6 включает в себя популярные возможности языка, продвижение которых продолжается до сих пор: операторы `let` и `const` для создания переменных, дефолтные значения параметров в определениях функций и стрелочные функции. Эта версия языка JavaScript отражает общую тенденцию к переходу на "современный" язык JavaScript, и потому под стандартом ES6 часто подразумевается "современный" язык JavaScript.

Все современные браузеры практически полностью поддерживают стандарт ES6 и уж точно поддерживают упомянутые популярные возможности. Вы можете без всякого риска программировать на JavaScript версии ES6, не используя компилятор для поддержки браузера.

Что такое ES9?

ES9 — это версия стандарта ECMAScript, опубликованная в 2018 году. На момент написания книги является последней версией стандарта.

Что такое ESNext?

ESNext — это название следующей версии стандарта ECMAScript, выпуск которой запланирован на 2019 год.

Что такое Ajax?

Ajax (Asynchronous JavaScript и XML, асинхронный JavaScript и XML) — это подход, подразумевающий использование JavaScript-кода для направления запросов на сервер после загрузки страницы. Раньше при этом возвращались XML-данные, которые обрабатывались браузером с помощью дополнительного JavaScript-кода. Сегодня мы обычно отправляем обратно данные в формате JSON или чистый HTML-код, непосредственно встраиваемый в приложение. В данной главе мы рассмотрим выполнение Ajax-вызова с помощью метода `ajax()` библиотеки jQuery, а также через новый API-интерфейс для WordPress с названием Heartbeat API.

Что такое JSON?

JSON (JavaScript Object Notation, объектная нотация JavaScript) — формат передачи данных, легко читаемый как человеком, так и компьютером. Такой формат особенно полезен при использовании JavaScript-кода, поскольку операторы, должным образом преобразованные в формат JSON, могут выполняться JavaScript-кодом без

какой-либо дополнительной обработки. Для работы с форматом JSON в PHP мы применяем функции `json_encode()` и `json_decode()`, входящие в состав ядра PHP, начиная с версии 5.2.

jQuery и WordPress

jQuery — это популярная JavaScript-библиотека, которая существенно упрощает работу с JavaScript-кодом. Так, например, библиотека jQuery упрощает выполнение Ajax-вызовов.

В установочный комплект WordPress входит собственная версия jQuery, которую можно использовать в панели администратора для написания скриптов, относящихся к пользовательскому интерфейсу и Ajax-запросам. Поскольку библиотека jQuery уже будет установлена на сервере, включить ее в клиентской части WordPress-приложения не составит большого труда.



На момент написания книги вместе с WordPress поставляется jQuery версии 1.12.4, в то время как последней версией jQuery является версия 3.3.1. Большинство тем и плагинов WordPress ориентировано на версию jQuery, поставляемую вместе с WordPress, и именно эту версию мы будем рассматривать здесь. Как показывает практика, переход на более новые версии jQuery представляет определенные сложности. Вы можете следить за ходом обсуждения данной проблемы на соответствующей странице форума, посвященного ядру WordPress ([oreil.ly/9j29R](https://wordpress.org/support/topic/jquery-version-1-12-4/)).

JavaScript-файл библиотеки jQuery расположен по адресу `/wp-includes/lib/js/jquery.js`. Обычно для загрузки jQuery нужно добавить в тег `<head>` веб-сайта ссылку следующего вида:

```
<script lang="JavaScript" src="/wp-includes/lib/js/jquery.js" />
```

Этот код сработает, если вы добавите его в файл `header.php` своей темы или с помощью хука `wp_head`, но, все же, будет правильнее, если для подключения файла `*.js` к своему WordPress-сайту вы воспользуетесь функцией `wp_enqueue_script()`. Добавьте строку `wp_enqueue_script('jquery');` в хук `init`, вызываемую из основного файла плагина, как показано здесь:

```
function sp_enqueue_scripts() {  
    wp_enqueue_script('jquery');  
}  
add_action('init', 'sp_enqueue_scripts')
```

В качестве первого параметра функции `wp_enqueue_script()` передается название подключаемого JavaScript-файла. Поскольку WordPress уже знает, какой файл называется `jquery`, и где он расположен³, для его подключения не нужны какие-либо другие параметры.

³ Узнать, какие из поставляемых JavaScript-файлов зарегистрированы в ядре WordPress, можно, взглянув на вызов функции `wp_default_scripts()` в файле `/wp-includes/script-loader.php`.

Подключение других JavaScript-библиотек

Для подключения других JavaScript-библиотек, которые еще неизвестны системе WordPress, необходимо передавать полный список параметров. В основной плагин приложения можно включить код приведенного далее примера, который будет загружать jQuery и любое количество необходимых JavaScript-библиотек. Хотя вы можете подключать свои скрипты с помощью хука `init`, лучше использовать для этого хуки `wp_enqueue_scripts` и `admin_enqueue_scripts`. Первый срабатывает перед подключением скриптов в клиентской части, а второй — перед подключением скриптов в панели администратора.

Пример

```
<?php
//JavaScript-код для клиентской части
function sp_wp_enqueue_scripts() {
    wp_enqueue_script('jquery');
    wp_enqueue_script(
        'schoolpress-plugin-frontend',
        plugins_url('js/frontend.js', __FILE__),
        array('jquery'),
        '1.0'
    );
}
add_action("wp_enqueue_scripts", "sp_wp_enqueue_scripts");

//JavaScript-код для панели администратора
function sp_admin_enqueue_scripts() {
    wp_enqueue_script(
        'schoolpress-plugin-admin',
        plugins_url('js/admin.js', __FILE__),
        array('jquery'),
        '1.0'
    );
}
add_action('admin_enqueue_scripts', 'sp_admin_enqueue_scripts');
?>
```

Использование хуков `wp_enqueue_scripts` и `admin_enqueue_scripts` позволяет загружать различные JavaScript-файлы в клиентской и серверной частях сайта. Вы можете добавить в этот код дополнительные проверки, чтобы загрузка jQuery выполнялась только на определенных страницах — это может уменьшить время загрузки тех страниц, которым не требуется jQuery. Распространенной практикой также является проверка атрибутов глобальной переменной `$post` или значений переменной `$_REQUEST`, используемых в панели администратора, таких как `$_REQUEST['page']` или `$_REQUEST['post_type']`.

Как вы помните, первый параметр функции `wp_enqueue_script()` — название скрипта. Второй параметр этой функции указывает расположение скрипта. Функция `plugins_url()` предназначена для определения URL-адреса относительно текущего файла `__FILE__`. Это сработает при включении данного кода в основной файл плагина. Передайте этой функции в качестве параметра выражение `dirname(__FILE__)`, если редактируемый вами файл находится в подкаталоге плагина.

Третий параметр функции `wp_enqueue_script()` позволяет указать зависимости скрипта. Мы передаем в качестве этого параметра выражение `array('jquery')` для скриптов `frontend.js` и `admin.js`, тем самым гарантируя, что перед их загрузкой будет загружена библиотека jQuery.

Где следует размещать кастомный JavaScript-код

Опять же, иногда нам придется подумать о том, где следует разместить определенный фрагмент кода. Куда его лучше добавить — в код темы или код плагина? Принимая решение о том, куда следует поместить тот или иной фрагмент JavaScript-кода, мы следуем общим правилам:

- ◆ Если код будет исполняться только один раз и, по большей части, является специфичным для использующей его страницы, то его можно включить непосредственно в код этой страницы в теге `<script>`.
- ◆ Если JavaScript-код исполняется более одного раза (т. е. является функцией или модулем), и связан с функциональностью или пользовательским интерфейсом темы, то его следует разместить в JavaScript-файле в каталоге темы (например, в файле `/themes/schoolpress/js/schoolpress.js`).
- ◆ Если JavaScript-код используется более одного раза на администраторских экранах приложения, то его следует разместить в файле `admin.js` в каталоге плагина (например, в файле `/plugins/schoolpress/js/admin.js`).
- ◆ Если JavaScript-код исполняется более одного раза в клиентской части приложения, но не является частью пользовательского интерфейса темы, то его следует разместить в файле `frontend.js` в каталоге плагина (например, в файле `/plugins/schoolpress/js/frontend.js`).
- ◆ Если выделение некоторого JavaScript-кода в отдельный файл, загружаемый только на определенных страницах, может обеспечить необходимый прирост производительности, то этот код следует разместить в отдельном JavaScript-файле⁴.
- ◆ Если вы применяете современные JavaScript-фреймворки или пишете на JavaScript значительную часть приложения, то нужно будет принять еще ряд других решений по организации JavaScript-файлов. Руководствуйтесь при этом своим здравым смыслом или соглашениями, принятыми в используемом вами фреймворке.

⁴ Самостоятельные JavaScript-файлы можно кэшировать или доставлять с использованием сети доставки контента (Content Delivery Network, CDN). JavaScript-код, встроенный в динамические PHP-файлы, гораздо труднее поддается кэшированию.

Эти правила отражают предпочитаемый нами стиль разработки и предлагаются лишь как один из возможных подходов. Некоторых разработчиков вгоняет в дрожь уже сама идея включения JavaScript-кода в теги `<script>` вместо размещения всего JavaScript-кода в файлах `*.js`.⁵ Сегодня в веб-разработке наблюдается тенденция к увеличению объема клиентского кода, и, соответственно, JavaScript-кода. Размещение всего JavaScript-кода в файлах `*.js` при этом обеспечивает безопасную разработку WordPress-приложений с поддержкой JavaScript.

Важно лишь иметь четкое представление о том, как организованы ваши JavaScript-файлы и код, чтобы исключить какие-либо затруднения с поддержкой сайта.

Ajax-вызовы в WordPress с использованием jQuery

Существует несколько способов осуществления Ajax-вызовов в WordPress. Фреймворк Vanilla JavaScript предлагает функцию `XMLHttpRequest()`, начиная с 2000 года, и вы по-прежнему можете использовать ее напрямую. Современные браузеры поддерживают такие новые возможности языка JavaScript, как `promise`, `async` и `await`, а также метод `fetch()`, каждая из которых может выполнять прилично выглядящие Ajax-вызовы с помощью "ванильного" JavaScript-кода. Angular, Vue.js, React и другие JavaScript-фреймворки предлагают собственные методы для выполнения Ajax-подобных вызовов. Наиболее популярный способ выполнения Ajax-вызовов в WordPress — использование для этой цели библиотеки jQuery. Этот способ демонстрируется в представленных далее примерах.

Для выполнения Ajax-вызовов в WordPress требуются два компонента: JavaScript-код в клиентской части, который будет запускать Ajax-запрос, и PHP-код в серверной части, который будет обрабатывать этот запрос и возвращать данные в формате HTML или JSON. Допустим, что вы хотите настроить страницу регистрации так, чтобы введенный логин автоматически проверялся на предмет того, не занят ли он уже другим пользователем. Если это так, можно выдать пользователю соответствующее сообщение с предложением изменить выбранный логин до того, как он нажмет на кнопку подтверждения. Тем самым мы сэкономим ему немного времени.

⁵ Сегодняшний Джейсон является одним из таких разработчиков. Если бы он мог вернуться в прошлое, то он попросил бы Джейсона из 2012 года проследить за тем, чтобы весь JavaScript-код плагина Paid Memberships Pro размещался в отдельных JS-файлах. Такой подход крайне полезен в случае плагинов и тем, распространяемых на неподконтрольных вам сайтах. При размещении JavaScript-кода в отдельных файлах владельцам сайтов легче оптимизировать его за счет объединения всего JavaScript-кода в одном файле или его обработки в "подвале", а не заголовке сайта. Размещение JavaScript-кода в отдельных файлах также позволяет избежать ситуации, когда сбой загрузки одного фрагмента встроенного в страницу JavaScript-кода не дает загрузиться другому фрагменту кода. Однако, несмотря на все сказанное, мы считаем вполне уместным размещение JavaScript-кода внутри PHP-файлов в тех случаях, когда вы имеете полный контроль над приложением и не хотите ничего усложнять.

Первое, что нужно для этого сделать, — добавить небольшой фрагмент JavaScript-кода в заголовок страниц, чтобы определить переменную `ajaxurl`. Это URL-адрес, на который будут отправляться все Ажак-запросы. Вот как это выглядит:

```
<script type="text/JavaScript">
var ajaxurl = '/wp-admin/admin-ajax.php';
</script>
```

В панели администратора WordPress этот скрипт будет встроенным по умолчанию. Однако для реализации Ажак-вызовов в клиентской части нам потребуется встроить его самим. Следующий пример иллюстрирует, как будет выглядеть код, определяющий переменную `ajaxurl` для клиентской части.

Пример

```
function my_wp_head_ajax_url()
{
?>
<script type="text/JavaScript">
var ajaxurl = '<?php echo admin_url("admin-ajax.php");?>';
</script>
<?php
}
add_action('wp_head', 'my_wp_head_ajax_url');
```

Теперь переменная `ajaxurl` будет доступна для остального JavaScript-кода клиентских страниц, и ее можно будет использовать в наших Ажак-вызовах. В нижней части страницы регистрации следует добавить JavaScript-код для проверки введенного логина, как показано в следующем примере.

Пример

```
<?php
//наш JavaScript-код для страницы
function my_wp_footer_registration_JavaScript()
{
    //проверяем, находимся ли мы на странице регистрации
    if(empty($_REQUEST['action']) || $_REQUEST['action'] != 'register')
        return;
?>
<script>
    //ожидаем, пока загрузится DOM-модель
    jQuery(document).ready(function() {
        //переменная для отслеживания времени ожидания
        var timer_checkUsername;

        //выявляем изменение поля user_login
        jQuery('#user_login').bind('keyup change', function() {
```

```

        // используем таймер для запуска проверки через 1 секунду
        // после прекращения ввода
        timer_checkUsername = setTimeout(function(){checkUsername();}, 1000);
    });
});

function checkUsername()
{
    //проверим, был ли введен логин
    var username = jQuery('#user_login').val();
    if(!username)
        return;

    //проверяем логин
    jQuery.ajax({
        url: ajaxurl,type:'GET',timeout:5000,
        dataType: 'html',
        data: "action=check_username&username="+username,
        error: function(xml){
            //время ожидания истекло, но не нужно пугать пользователя
        },
        success: function(response){
            //скрываем отображенные нами флаги
            jQuery('#username_check').remove();

            //отображаем, подходит ли логин (1), или он уже занят (0)
            if(response == 1)
                jQuery('#user_login').after(
                    '<span id="username_check" class="okay">Okay</span>'
                );
            else
                jQuery('#user_login').after(
                    '<span id="username_check" class="taken">Taken</span>'
                );
        }
    });
}
}
</script>
<?php
}
add_action('wp_footer', 'my_wp_footer_registration_JavaScript');
?>

```

Приведенный в этом примере код привязывается к хуку `wp_footer`, чтобы JavaScript-код добавлялся в конец HTML-вывода. Сначала мы проверяем, выполняется ли равенство `$_REQUEST['action'] == "register"`, чтобы убедиться, что мы находимся на дефолтной странице регистрации WordPress.

Если ваш плагин имеет собственную страницу регистрации подобно плагину Paid Memberships Pro, то, чтобы выяснить, на какой странице вы находитесь, используйте проверку вида `if(!is_page("membership-checkout"))`. Также не забудьте заменить в своем JavaScript-коде `#user_login` на идентификатор, присвоенный полю для ввода логина на странице регистрации.

В данном коде мы задействуем метод `jQuery(document).ready()` для регистрации момента завершения загрузки DOM-модели, после чего вызываем метод `jQuery('#user_login').bind('keyup change', ...)` для выявления изменений, вносимых пользователем в поле с помощью клавиатуры или каким-либо иным образом. При изменении поля мы используем функцию `setTimeout()`, чтобы запустить проверку логина с задержкой в одну секунду. Если пользователь возобновит ввод до истечения времени ожидания, таймер будет сброшен и отсчет одной секунды будет произведен еще раз. В результате функция `checkUsername()` будет запущена через одну секунду после того, как пользователь завершит изменение поля.

Внутри функции `checkUsername()` мы вызываем метод `jQuery.ajax()`. Однако перед тем, как это сделать, мы проверяем, содержит ли значение поле для ввода логина. Внутри метода `jQuery.ajax()` мы задаем в качестве URL-адреса значение переменной `ajaxurl`, которая к этому моменту уже должна быть задана посредством хука `wp_head`.

В качестве типа вызова мы назначаем метод `GET`. Также возможен метод `POST`. Доступны и методы `DELETE` и `PUT`, но их поддерживают не все браузеры. Принимая решение о том, какой метод запроса следует выбрать для Ajax-вызова, руководствуйтесь той же логикой, что и в случае выбора метода запроса при отправке форм. Если вы "получаете" данные, как это делается в данном примере, лучше будет метод `GET`. Если же вы отправляете данные для их сохранения, можно использовать метод `POST`.

Мы задаем здесь время ожидания равным 5000 (5 секунд). По истечении этого времени запрос будет отменен, и будет запущено действие, определенное на случай ошибки. Величину времени ожидания следует выбирать, исходя из того, сколько времени может потребоваться серверу на обработку конкретного запроса. Слишком малое время ожидания может вызвать преждевременную отмену запросов. Слишком большое время ожидания может привести к тому, что пользователь будет долго ожидать выполнения запроса, который, возможно "завис" на стороне сервера.

В качестве типа данных мы задаем здесь значение `html`. Тем самым мы даем указание библиотеке jQuery поместить результат в строку. Если указать тип данных `json`, то получаемый результат будет помещаться в объектную переменную языка JavaScript. Имеется и ряд других типов данных, включая `xml`, `jsonp`, `script` и `text`. В документации по jQuery можно найти сведения о том, когда следует использовать каждый из этих типов данных, и как их обрабатывает jQuery.

В качестве данных мы задаем выражение `"action=check_username&username="+username`, которое будет передавать определенное нами действие и логин в качестве параметров скрипту `wp-admin-ajax.php` и нашей программе на стороне сервера.

Затем мы задаем обработчики на случай ошибки и на случай успеха. При появлении ошибки вы могли бы уведомить о ней пользователя, но, поскольку данная функция не является критически важной, мы ничего здесь не делаем. В случае успеха мы удаляем старый элемент `#username_check` и добавляем сообщение "Okay" или "Taken" после поля для ввода логина.



Полную API-документацию по методу `jQuery.ajax()` можно найти на веб-сайте библиотеки jQuery (bwwwp.com/ajax-jq).

Теперь давайте взглянем на код серверной части. Следующий пример иллюстрирует, как будет выглядеть код, который вы должны разместить в файле `functions.php`, своем кастомном плагине или файле `*.php` внутри каталога `/services/` плагина для прослушивания Ajax-запроса и возвращения назад значения 1 или 0 в зависимости от того, доступен логин или нет.

Пример

```
<?php
//выявляем Ajax-запрос на действие check_username
function wp_ajax_check_username() {
    global $wpdb;
    $username = $_REQUEST['username'];

    $taken = username_exists($username);

    if ($taken)
        echo "0";      // занят
    else
        echo "1";      // доступен
}
add_action('wp_ajax_check_username', 'wp_ajax_check_username');
add_action('wp_ajax_nopriv_check_username', 'wp_ajax_check_username');
?>
```



Функция `username_exists()` возвращает идентификатор пользователя, если указанный пользователь существует, или значение `false`, если это не так. Документацию по этой функции можно найти в Кодексе WordPress (oreil.ly/Rlgf7).

WordPress предлагает два хука для определения Ajax-функций. В данном примере задействованы оба хука:

- ◆ `wp_ajax_{action}` — запускается для авторизованных пользователей.
- ◆ `wp_ajax_nopriv_{action}` — запускается для неавторизованных пользователей.

Поскольку на странице регистрации пользователи по определению не являются авторизованными, нам требуется хук `wp_ajax_nopriv_`. Но поскольку данная проверка

также может выполняться на экране добавления нового пользователя в панели администратора, мы также с помощью хука `wp_ajax_` обрабатываем эту ситуацию.

Если ваш Ajax-сервис предназначен только для авторизованных пользователей, оставьте только хук `wp_ajax_`. Если же вам нужно, чтобы Ajax-сервис был доступен и для авторизованных, и для неавторизованных пользователей, используйте оба хука.

Также обратите внимание, как в определении действий имя нужного нам параметра `action (check_username)` добавляется к имени хука. Такой хук сработает только в том случае, если `$_REQUEST['action'] == "check_username"`.

Управление количеством Ajax-запросов

При использовании Ajax-запросов важно отслеживать их количество. В противном случае вы рискуете подвергнуть повышенной нагрузке сервер и клиентский браузер, что может привести к "зависанию" сеанса пользователя или всего сайта.

Так, в предыдущем примере кода мы запускаем Ajax-запрос для проверки доступности логина, выждав одну секунду после обновления содержимого поля для ввода логина. Однако после отправки запроса пользователь может возобновить ввод, что приведет к запуску еще одного Ajax-запроса. Если ваш сервер не успевает обработать запрос за одну секунду, запросы начнут накапливаться.

Конечно, наш код для проверки логина вряд ли когда-либо "пойдет в разнос", однако во многих случаях такое вполне возможно. Простейший пример — запуск Ajax-запроса при щелчке по кнопке. Если пользователь щелкнет по кнопке 20 раз, то на сервер будет отправлено 20 запросов. Так что держите количество запросов под контролем.

В общем случае, управлять количеством Ajax-запросов можно с помощью одного из следующих двух подходов:

- ◆ Не допускать отправку пользователем запроса, пока не будет обработан предыдущий запрос того же типа.
- ◆ Отменять предыдущий запрос того же типа в случае отправки нового запроса.

Какой из этих вариантов лучше выбрать, зависит от того, что делает Ajax-запрос. Обычно, если вы "получаете" данные, то лучше отменять предыдущие запросы, заменяя их более "свежим" запросом. Если же вы "постите" данные, то лучше игнорировать новый запрос, пока не завершится обработка старого запроса.

В зависимости от того, что представляют собой приложение и запрос, возможны разные способы отключения или отмены запросов. Поскольку и в случае успеха, и в случае ошибки внутри метода `ajax()` библиотеки jQuery вызывается функция обратного вызова `complete`, вы можете использовать ее для того, чтобы снова включить кнопку или другой элемент, предназначенный для запуска конкретного Ajax-запроса, как показано в следующем примере.

//Вариант 1: Отключение кнопки на время обработки Ajax-запроса

```
jQuery('#button').click(function() {
    //отключаем кнопку
    jQuery(this).attr('disabled', 'disabled');

    //выполняем Ajax-запрос
    jQuery.ajax({
        url: ajaxurl,type:'GET',timeout:5000,
        dataType: 'html',
        error: function(xml){
            //действия в случае ошибки
        },
        success: function(response){
            //действия в случае успеха
        }
        complete: function() {
            //снова включаем кнопку
            jQuery('#button').removeAttr('disabled');
        }
    });
});
```

Сходным образом можно выполнять и отмену старого запроса при поступлении нового запроса:

//Вариант 2: Отмена старого запроса при поступлении нового

```
var ajax_request;
jQuery('#button').click(function() {
    //отменяем все имеющиеся запросы
    if(typeof ajax_request !== 'undefined')
        ajax_request.abort();

    //выполняем Ajax-запрос
    ajax_request = jQuery.ajax({
        url: ajaxurl,type:'GET',timeout:5000,
        dataType: 'html',
        error: function(xml){
            //действия в случае ошибки
        },
        success: function(response){
            //действия в случае успеха
        }
    });
});
```

Heartbeat API

В одном из предыдущих примеров этой главы мы создали Ajax-вызов, срабатывающий при обновлении содержимого одного из полей формы. Однако иногда требуется, чтобы во время работы веб-приложения определенные обновления периодически выполнялись сами по себе. Например, иногда нужно отслеживать появление новых комментариев на дискуссионном форуме и автоматически извлекать свежие комментарии по мере их добавления. При использовании JavaScript-кода это обычно делается путем опроса серверной части с интервалом в несколько секунд с помощью Ajax-вызова, запускаемого функцией `setInterval()`. Еще один способ состоит в том, чтобы воспользоваться API-интерфейсом для WordPress с названием Heartbeat API.

Heartbeat API, доступный в WordPress, начиная с версии 3.6, может существенно упростить для вас задачу выполнения в приложении обновлений в режиме квазиреального времени. При этом с интервалом в 15 секунд (или менее, если изменить настройки), приложение отправляет тактовый запрос с клиента на сервер и обратно. Во время этого путешествия туда и обратно, вы можете выполнять такие вещи, как автоматическое сохранение состояния приложения или загрузка свежего контента. В WordPress 3.6 интерфейс Heartbeat API служит для автоматического сохранения и блокировки постов, и также выдачи предупреждений об истечении срока регистрации. В этом разделе мы поговорим о том, как вы можете использовать Heartbeat API в своем приложении.

На первый взгляд может показаться, что Heartbeat API представляет собой нечто очень сложное, но на самом деле это просто набор данных, передаваемых с клиента на сервер и обратно посредством периодических Ajax-вызовов. Через хуки вы можете получить доступ к тем данным, которые передаются на сервер или принимаются от него, чтобы получить или отправить нужные вам данные.

Далее представлен простейший пример использования Heartbeat API. Этот код делает всего одну вещь — отправляет на сервер сообщение `marco`. Обнаружив это сообщение, сервер возвращает клиенту сообщение `polo`. Оба сообщения протоколируются в консоли JavaScript; т. е. с интервалом в 15 секунд в консоль выводится следующее:

```
Client: marco
```

```
Server: polo
```

Использование Heartbeat API включает в себя три компонента: инициализацию, клиентский JavaScript-код и серверный PHP-код.

Инициализация

Пример инициализации

```
//подключаем heartbeat.js и наш JavaScript-код
function hbdemo_init()
{
```

```

/*
    //добавьте здесь свои условные конструкции,
    //чтобы этот код выполнялся на нужных вам страницах, например:
        if(is_admin())
            return;    // исключаем запуск кода в панели
администратора
*/

//подключаем Heartbeat API
wp_enqueue_script('heartbeat');

//загружаем наш JavaScript-код в подвале
add_action("wp_footer", "hbdemo_wp_footer");
}
add_action('init', 'hbdemo_init');

```

Эта первая функция подключает файл `heartbeat.js` и определяет действие, помещающее наш JavaScript-код в "подвале" с помощью хука `wp_footer`. Если вам нужно, чтобы этот код тактового запроса запускался только на определенных страницах (что весьма вероятно), вам следует разместить здесь соответствующие проверки.

Клиентский JavaScript-код

Пример JavaScript-кода

```

<?php
//наш JavaScript-код для отправки/обработки на клиентской стороне
function hbdemo_wp_footer()
{
    ?>
<script>
    jQuery(document).ready(function() {
        //подключение к событию heartbeat-send: клиент отправляет сообщение
        //'marco' в переменной 'client' внутри массива данных
        jQuery(document).on('heartbeat-send', function(e, data) {
            console.log('Client: marco');

            //данные, необходимые для запуска Ajax-вызова
            data['client'] = 'marco';
        });

        //подключение к событию heartbeat-tick: клиент ищет переменную 'server'
        //в массиве данных и протоколирует ее в консоли
        jQuery(document).on('heartbeat-tick', function(e, data) {
            if(data['server'])
                console.log('Server: ' + data['server']);
        });
    });

```



```

//подключение к событию heartbeat-error для протоколирования ошибок
jQuery(document).on('heartbeat-error',
    function(e, jqXHR, textStatus, error) {
        console.log('BEGIN ERROR');
        console.log(textStatus);
        console.log(error);
        console.log('END ERROR');
    });
});
</script>
<?php
}
?>

```

Эта вторая функция помещает наш JavaScript-код в "подвал". В JavaScript-коде мы используем метод `jQuery(document).ready()` для запуска нашего кода после загрузки DOM-модели. Затем мы подключаемся к трем событиям JavaScript, запускаемым интерфейсом Heartbeat API:

- ◆ Событие `heartbeat-send` срабатывает непосредственно перед отправкой на сервер данных тактового запроса. Чтобы отправить собственные данные, необходимо добавить значение в массив `data`, передаваемый посредством этого события.
- ◆ Событие `heartbeat-tick` срабатывает, когда сервер дает ответ. Чтобы посмотреть, какие данные прислал сервер, необходимо проверить содержимое массива `data`, передаваемого посредством этого события.
- ◆ Событие `heartbeat-error` срабатывает в случае ошибки при выполнении метода `jQuery.ajax()`, используемого для отправки данных на сервер. Здесь можно разместить код для отладки или аккуратной обработки в том случае, если Ajax-вызовы откажутся работать в вашем эксплуатационном окружении.

Серверный PHP-код

Пример серверного кода PHP

```

//обработка сообщения на сервере
function hbdemo_heartbeat_received($response, $data)
{
    if($data['client'] == 'marco')
        $response['server'] = 'polo';

    return $response;
}
add_filter('heartbeat_received', 'hbdemo_heartbeat_received', 10, 2);

```

Представленная здесь третья РНР-функция запускается хуком `heartbeat_received` и обрабатывает полученные от клиента данные. Чтобы добавить собственные данные для отправления клиенту, необходимо обновить содержимое переменной `response`.

Теперь давайте рассмотрим более реалистичный пример. На странице заданий приложения SchoolPress есть раздел с данными о количестве присланных и еще ожидаемых заданий. Давайте применим Heartbeat API для обновления этих цифр при поступлении новых заданий.

В нашем шаблоне количество заданий будет отображаться примерно так, как показано в следующем примере.

Пример

```
?>
<div>
    Submitted:
    <span id="assignment_count">
        <?php echo count($assignment->submissions);?>
    </span>
    /
    <?php echo count($course->students);?>
</div>
<?php
```

Инициализация

Пример инициализации

```
function sp_init_assignments_heartbeat()
{
    //игнорируем, если мы не на странице заданий
    if(strpos($_SERVER['REQUEST_URI'], "/assignment/") === false)
        return;

    //подключаем Heartbeat API
    wp_enqueue_script('heartbeat');

    //загружаем наш JavaScript-код в подвале
    add_action("wp_footer", "sp_wp_footer_assignments_heartbeat");
}
add_action('init', 'sp_init_assignments_heartbeat');
```

Пока все делается почти также, как в предыдущем простейшем примере. Мы лишь обеспечиваем запуск этого кода исключительно на странице заданий, проверяя наличие сегмента `/assignment/` в URI-адресе.

Клиентский JavaScript-код

Пример JavaScript-кода

```
<?php
function sp_wp_footer_assignments_heartbeat()
{
    global $post;    //пост для текущего задания
?>
<script>
jQuery(document).ready(function() {
    //событие heartbeat-send
    jQuery(document).on('heartbeat-send', function(e, data) {
        //проверяем, есть ли у нас массив для данных приложения SchoolPress
        if(!data['schoolpress'])
            data['schoolpress'] = new Array();

        //отправляем на сервер post_id этого задания и текущее количество заданий
        data['schoolpress']['assignment_post_id'] = '<?php echo $post->ID;?>';
        data['schoolpress']['assignment_count'] = jQuery('#assignment-count').val();
    });

    //событие heartbeat-tick
    jQuery(document).on('heartbeat-tick', function(e, data) {
        //обновляем количество заданий
        if(data['schoolpress']['assignment_count'])
            jQuery('#assignment-count').val(data['schoolpress']['assignment_count']);
    });
});
</script>
<?php
}
?>
```

Обратите внимание, что мы сохраняем свои данные в массиве `schoolpress` внутри массива `data`. Все данные, которые имеют отношение к Heartbeat API, мы будем сохранять в этом массиве как в своего рода пространстве имен, чтобы имена наших переменных не конфликтовали с именами переменных других плагинов, также использующих Heartbeat API. С каждым тактовым запросом мы отправляем на сервер идентификатор поста задания и текущее количество заданий.



Важно, чтобы с тактовым запросом на сервер отправлялось хотя бы *что-то*. Тактовый запрос не будет отправляться на сервер при отсутствии отправляемых данных.

Серверный PHP-код

Пример серверного PHP-кода

```
//обработка сообщения на сервере
function sp_heartbeat_received_assignment_count($response, $data)
{
    //проверяем наличие значения assignment_post_id
    if(!empty($data['schoolpress']['assignment_post_id']))
    {
        $assignment = new Assignment(
            $data['schoolpress']['assignment_post_id']
        );
        $response['schoolpress']['assignment_count'] = count(
            $assignment->submissions
        );
    }

    return $response;
}

add_filter('heartbeat_received',
    'sp_heartbeat_received_assignment_count', 10, 2);
```

Здесь мы проверяем наличие значения `assignment_post_id` в переданных клиентом данных. Если это значение присутствует, то мы загружаем задание и возвращаем количество присланных заданий в виде значения `assignment_count`, наличие которого будет проверять наш клиентский JavaScript-код.

Этот код можно модифицировать так, чтобы он выявлял изменение количества заданий (путем сравнения присланного клиентом значения с значением, сохраненным на сервере) и в таком случае отправлял обратно сообщение, уведомляющее учителя о необходимости обновить страницу для просмотра вновь присланных заданий. Также можно отправлять непосредственно данные о новых заданиях и помещать их в отображаемый на странице список.

Наконец, при желании можно уменьшить или увеличить величину тактового промежутка, переопределив настройки с помощью следующего кода:

```
function sp_heartbeat_settings($settings = array())
{
    $settings['interval'] = 20; //20 секунд вместо дефолтных 15 секунд
    return $settings;
}

add_filter('heartbeat_settings', 'sp_heartbeat_settings');
```

Обратите внимание, что на момент написания книги данный API позволял использовать только значение в диапазоне от 15 до 60 секунд. При выходе за нижнюю или верхнюю границу этого диапазона величина такта будет задана равной, соответст-

венно, 15 или 60 секундам. Наложение такого ограничения в Heartbeat API вполне оправдано, поскольку в любой момент времени один и тот же тактовый запрос может быть задействован множеством плагинов и процессов. Если определенный опрос нужно выполнять чаще, чем через 15 секунд, определите его как отдельный Аякс-вызов, запускаемый собственными вызовами функции `setInterval` или `setTimeout` в JavaScript-коде.

Heartbeat API можно рассматривать как стандартное средство для отправки запросов между клиентом и сервером. Если вы имеете дело с каким-либо нестандартным случаем (к каковым вполне можно отнести опрос сервера с интервалом в одну секунду), то вам следует реализовать собственную систему наподобие Heartbeat API.

Ограничения WordPress в плане асинхронной обработки

Большинство приложений WordPress выполняет PHP-скрипты, используя сервер Apache или NGINX. При надлежащей оптимизации такая конфигурация позволяет одновременно обслуживать много небольших соединений, что хорошо подходит для асинхронных JavaScript-приложений. Однако как из-за ограничений этих серверов, так и в силу общей нагрузки WordPress за счет серверных вызовов WordPress-сервис, запущенный на сервере Apache или NGINX, никогда не будет столь же быстрым, как небольшой JavaScript-сервис, запущенный на такой платформе, как Node.js, созданной специально для обработки асинхронных JavaScript-вызовов.

Несмотря на все сказанное, вы вполне можете добиться от системы WordPress и стоящей за ней архитектуры достойных результатов. Мы рекомендуем сначала всегда создавать приложение наиболее очевидным образом, а затем выборочно извлекать отдельные компоненты для масштабирования по мере повышения требований к производительности.

База пользователей вашего приложения включает в себя лишь 30 сотрудников вашей компании? Тогда вы вполне можете обойтись возможностями WordPress в плане выполнения JavaScript-кода в режиме реального времени.

Вы ожидаете, что у вашего приложения будут тысячи пользователей, одновременно реализующих десятки соединений? Что ж, тогда вам потребуется производительное аппаратное обеспечение, но, вероятно, вы также сможете обойтись возможностями WordPress.

Вы ожидаете, что у вашего приложения будут миллионы пользователей, одновременно выполняющих десятки тысяч соединений? Если это так, то вам потребуется команда инженеров с высоким уровнем квалификации, и будем надеяться, что у вас есть деньги на оплату их услуг. При этом вам в любом случае потребуется либо задействовать WordPress на пределе возможностей, либо воспользоваться другими платформами для обработки взаимодействий в режиме реального времени.

О подобном масштабировании мы подробно поговорим в *главе 14*.

JavaScript-фреймворки

Одно из препятствий для перехода на использование JavaScript-кода состоит в том, что все наши удобные функции и структуры данных являются нативными функциями и структурами данных языка PHP. По мере того, как все бóльшая часть разработки на платформе WordPress начинает приходиться на JavaScript-код, становится все более актуальной потребность в некотором фреймворке, способном упростить организацию JavaScript-разработки.

Backbone.js

Backbone.js — это JavaScript-фреймворк, включающий в себя модели, представления и коллекции моделей. Такая конфигурация во многом аналогична MVC-фреймворкам, применяемым для разработки серверного PHP-кода. В случае традиционных MVC-фреймворков под буквой "С" подразумевается выполняющий управление контроллер. В случае Backbone.js управление приложением осуществляется внутри представлений и за пределами JavaScript-фреймворка.

Фреймворк Backbone.js был добавлен в ядро WordPress в версии 3.5, и с тех пор активно задействован в обновлениях медиабιβотеки и кастомайзера тем. Как разработчики ядра WordPress, так и большинство разработчиков WordPress-приложений либо перешли на использование фреймворка Backbone.js, либо "перескочили" через него сразу на использование более новых библиотек и фреймворков JavaScript.

При этом важно понимать, как можно использовать Backbone.js для JavaScript-разработки на платформе WordPress. Если вы будете посредством Backbone.js отображать клиентскую часть приложения, то главной точкой пересечения с WordPress будет момент сохранения коллекций и моделей в базе данных с помощью серверной части приложения.

Допустим, что на сайте SchoolPress имеется интерфейс для добавления групп студентов к некоторому заданию. Этот интерфейс может включать в себя поле для ввода имени группы и кнопку для добавления группы Add Group (Добавить группу). При описанной в этой главе традиционной схеме применения Ajax-запросов последовательность событий будет выглядеть следующим образом:

1. Пользователь вводит новое имя группы.
2. Пользователь щелкает по кнопке Add Group (Добавить группу).
3. Имя группы отправляется на сервер посредством Ajax-вызова.
4. Сервер (WordPress) обрабатывает это имя, добавляет новую группу и возвращает некоторые данные.
5. Клиент использует JavaScript-код для синтаксического разбора ответа и обновления списка групп в клиентской части приложения.

В случае приложения на базе Backbone.js вы будете более точно зеркалировать список групп в модели и коллекции, определенных в JavaScript-коде. При этом

схема работы может быть такой же, как и в случае обычного Ajax-приложения, однако более подходящей в таком случае будет следующая схема работы:

1. Пользователь вводит новое имя группы.
2. Пользователь щелкает по кнопке Add Group (Добавить группу).
3. Имя группы используется для создания нового экземпляра модели группы и добавляется в коллекцию групп в Backbone.js.
4. Коллекция кодируется таким образом, чтобы при каждом ее изменении производилось обновление сервера (WordPress) посредством Ajax-вызова.
5. Представление текущей коллекции групп отправляется на сервер.
6. WordPress обновляет внутреннее представление коллекции в базе данных в соответствии с присланным представлением.

Таким образом, вместо того, чтобы сначала производить обновление в серверной части, а затем принимать от нее указание о том, как должна выглядеть клиентская часть, при использовании Backbone.js мы сначала обновляем клиентскую часть, а затем клиентская часть сообщает серверной части, как следует сохранить данные.

По адресу bwawwp.com/backbonejs-example/ вы найдете пример, демонстрирующий реализацию некоторых функций приложения SchoolPress и с помощью традиционной технологии Ajax, и с помощью Backbone.js.

Вот ресурсы, содержащие дополнительную информацию о фреймворке Backbone.js и его использовании в сочетании с WordPress:

- ◆ Официальный сайт фреймворка Backbone.js (backbonejs.org).
- ◆ Ресурсы по Backbone.js и WordPress Resources от Питера Р. Найта (bwawwp.com/bb-wp-knight).

React

React — это JavaScript-библиотека для создания интерактивных пользовательских интерфейсов. В отличие от многих других описываемых здесь библиотек, фреймворк React не предназначен для создания всего приложения и включает в себя лишь ту составляющую типичного MVC-фреймворка, которая относится к представлениям.

При этом вы определяете представления для каждого состояния приложения, после чего React отслеживает состояние приложения и обновляет пользовательский интерфейс по мере изменения данных. Например, счетчик общего количества пользователей на сайте (созданный с надлежащим применением фреймворка React) автоматически обновится, если вы добавите нового пользователя на этом же экране или если новый пользователь зарегистрируется сам на своем устройстве.

В React реализован компонентный подход. Компонент может представлять собой нечто очень простое вроде отдельного текстового поля или кнопки. Из нескольких подкомпонентов могут создаваться более сложные компоненты. Со многими из имеющихся в WordPress компонентов React можно ознакомиться в GitHub-

репозитории редактора Gutenberg (oreil.ly/GDqL4). Каждый из таких модулей WordPress, как данный редактор, обладает собственной библиотекой вспомогательных компонентов (oreil.ly/daXvW).

React также позволяет создавать нативные мобильные приложения с помощью соответственно названной библиотеки React Native. В отличие от обычных веб-приложений, которые пишутся один раз для использования на любых устройствах, эти "нативные" приложения предназначены только для мобильных устройств. Один и тот же React-код не пригоден и для веб-приложения, и для мобильного приложения. В то же время, один и тот же язык программирования JavaScript концепции фреймворка React подойдет для создания пользовательского интерфейса как для веб-приложения, так и для мобильного приложения, в зависимости от того, что вам нужно.

Пользовательский интерфейс и компоненты нового блочного редактора были созданы с помощью фреймворка React. В *главе 11* мы рассмотрим способы создания и расширения блоков, включая рекомендуемую схему JavaScript-разработки, способную усилить некоторые из затронутых в этой главе концепций. А до этого момента мы рекомендуем вам ознакомиться с REST API для WordPress, которому целиком посвящена *глава 10*.

REST API в WordPress

С выходом версии 4.4 системы WordPress в проект был добавлен API-интерфейс REST API, который существенно упростил расширение данной CMS. WordPress REST API позволяет разработчикам отправлять данные системе WordPress и получать данные от нее, действуя из-за ее пределов. Это делает возможным создание разнообразных удобных приложений с привлекательным интерфейсом, которые могут не использовать традиционный пользовательский интерфейс WordPress, будучи разработанными на базе данной CMS или интегрированными с ней. Благодаря этому вы можете обмениваться данными с другими приложениями независимо от того, на каком языке они написаны. Прежде чем мы углубимся в специфику работы с WordPress REST API, давайте кратко коснемся того, что обычно понимается под "REST API".



Данная глава во многом повторяет содержание подробного руководства по использованию REST API в WordPress (oreil.ly/400J5). В некоторых местах мы излагаем материал более подробно или приводим примеры, несущие больший смысл в контексте данной книги. Обратите внимание, что те вещи, которые мы описываем здесь кратко, могут быть рассмотрены более подробно в руководстве по WordPress.

Что такое REST API?

Возможно, вы уже знаете, что такое REST API, и даже удивлены, что этот API не был встроен в WordPress раньше. Давайте попробуем разобраться с тем, что означают слова "WordPress REST API" так, чтобы было понятно даже ребенку.



WordPress REST API иногда также называют "JSON REST API". В данной главе мы опустим слово "WordPress" и будем писать просто "REST API".

Давайте рассмотрим некоторые основные понятия.

API

Прикладной программный интерфейс (Application Programming Interface, API) — это набор функций и инструментов, встраиваемых в программное приложение и используемых разработчиками для взаимодействия с этим программным приложением изнутри или извне; причем внешнее взаимодействие обычно представляет собой взаимодействие веб-приложений посредством Интернета. Применяя различные способы настройки, вы можете с помощью API либо отправлять данные или

контент другому приложению, либо получать их, таким образом обеспечивая эффективную интеграцию двух или более приложений.

REST API позволяет вам взаимодействовать с базой данных WordPress из-за пределов своего WordPress-сайта.

REST

REST (Representational State Transfer, передача состояния представления) — это архитектурный стиль для определения HTTP-коммуникации между веб-приложениями. Существует ряд руководящих принципов для создания REST API, и если интерфейс удовлетворяет каждое из следующих ограничений, его называют "RESTful API":

- ◆ клиент-серверная архитектура;
- ◆ отсутствие состояния;
- ◆ кэшируемость;
- ◆ единообразие интерфейса;
- ◆ многослойная структура;
- ◆ код по требованию.

JSON

Как уже упоминалось ранее, JSON — это открытый стандарт текстового формата для отправки читаемого текста в таких объектах данных, как массивы и пары "имя—значение". Ответ на HTTP-запрос к REST API должен быть представлен в формате JSON. Вы должны быть в состоянии произвести синтаксический разбор данных в формате JSON, содержащихся в HTTP-ответе, и использовать эти данные в своем приложении.

HTTP

HTTP — это стандартный протокол передачи запросов и ответов между клиентом и сервером. Каждый раз, когда вы посещаете какой-либо веб-сайт в своем браузере, вы отправляете HTTP-запрос и получаете HTTP-ответ в виде загружаемой браузером веб-страницы. Обмен данными с WordPress посредством REST API осуществляется по такому же протоколу, однако вместо веб-страницы в данном случае возвращаются необработанные данные в формате JSON. WordPress по умолчанию позволяет вам отправлять HTTP-запросы для получения доступа к такому публичному контенту, как посты и страницы, в формате JSON, для чего нужно ввести в адресной строке браузера соответствующий URL-адрес или, иначе говоря, *маршрут API*.

В таких RESTful API, какой используется в WordPress, возможны четыре основных типа HTTP-запроса:

- ◆ POST ◆ PUT
- ◆ GET ◆ DELETE

Эти методы запроса также иногда называют "CRUD-методами" (Create, Read, Update, Delete — создание, чтение, обновление, удаление). Как бы вы ни называли эти методы с очевидным смыслом, они применяются путем отправки HTTP-запроса на выполнение соответствующего действия и последующего получения результатов этого действия в HTTP-ответе.

Попробуйте добавить сегмент `/wp-json/wp/v2/posts` после доменного имени вашего сайта на базе WordPress, и вы увидите содержимое ваших постов в формате JSON. Это пример метода HTTP-запроса `GET` и его ответа. Процесс отправки HTTP-запроса и получения HTTP-ответа называют передачей "*HTTP-сообщений*".

Каждое HTTP-сообщение состоит из трех частей: запроса, заголовков и тела сообщения.

Запрос

Запрос (или *строка запроса*) отправляется с клиента на сервер и включает в себя метод запроса, URL-адрес запроса и версию HTTP:

- ♦ Метод запроса — `GET`, `POST`, `PUT`, `DELETE`.
- ♦ URL-адрес запроса — URL-адрес маршрута API.
- ♦ Версия HTTP — версия протокола HTTP.

Вместе эти элементы выглядят так:

```
GET URL HTTP/1.1
```

Строка состояния возвращается в ответе на запрос и включает в себя версию HTTP и код состояния:

```
HTTP/1.1 200 OK
```

Поиск в Интернете информацию по ключевой фразе "Коды состояния HTTP и что они означают", вы можете ознакомиться с полным списком кодов состояния HTTP, возвращаемых в HTTP-ответах. Чтобы действительно понимать, что представляет собой протокол HTTP, рекомендуем вам прочитать книгу **HTTP: The Definitive Guide** (O'Reilly) под авторством Брайана Тотти, Дэвида Гурли, Марджори Сэйер, Аншу Аггарвала и Сайлу Редди.

Процесс передачи запроса серверу и возвращения ответа клиенту показан на рис. 10.1.



Рис. 10.1. Строка запроса

Заголовки

Отправляемый в HTTP-сообщении HTTP-заголовок содержит метаданные, которые могут передаваться в HTTP-запросе и возвращаться в HTTP-ответе. В заголовке можно использовать множество стандартных полей запроса и ответа, а также кастомные поля заголовка. Обычно заголовки служат для передачи с клиента на сервер и обратно такой информации, как сведения об аутентификации и разрешениях. Полный список стандартных полей заголовка для запроса и ответа можно найти в Википедии (oreil.ly/cG6hn).



Рис. 10.2. Использование инструментов разработчика браузера Chrome для просмотра заголовков веб-страницы

В Chrome и других браузерах на движке Webkit можно просмотреть заголовки текущей страницы и выполняемых вызовов API. Для этого выберите в меню пункт **Developer Tools** (Инструменты разработчика) и откройте вкладку **Network** (Сеть).

В списке слева выберите документ или Ajax-вызов, а затем откройте вкладку **Headers** (Заголовки). На рис. 10.2 показано, как выглядят заголовки страницы Википедии, ссылку на которую мы только что приводили.



Один из наиболее важных заголовков — Status Code (Код состояния). Загруженная без ошибок веб-страница обычно имеет код состояния 200 "OK". Однако показанная на рис. 10.2 страница Википедии имеет код состояния 304 "Not Modified". Дело в том, что Википедия использует схему кэширования в браузере, позволяющую экономить ресурсы за счет того, что обновляемая пользователем страница передается только в том случае, если ее содержимое изменилось. В *главе 14* мы обсудим, как можно настроить кэширование в браузере с помощью плагина W3 Total Cache.

Тело сообщения

Тело сообщения требуется не всегда, но может передаваться с запросом или ответом. REST API обычно возвращает в теле сообщения JSON-объект с данными WordPress, запрошенными вами в HTTP-запросе. В зависимости от используемого метода запроса, получаемый вами ответ может не содержать тело сообщения.

Зачем нужен REST API в WordPress?

Что ж, REST API позволяет вам делать почти все то, что вы можете делать из обычной панели администратора WordPress, из любого другого места, являясь при этом стандартным и безопасным способом выполнения этих действий. Далее приведены примеры возможных применений интерфейса REST API.

- ♦ Предоставление общего доступа к данным WordPress — в последних версиях WordPress данный API активируется по умолчанию, что позволяет легко предоставлять доступ любому пользователю к любому общедоступному контенту вашего приложения, такому как посты, страницы, медиаданные и комментарии. Хотя кастомные типы постов не являются общедоступными по умолчанию, с помощью REST API можно сделать общедоступными любой такой тип и любые его данные. Вы также можете создать общедоступные кастомные конечные точки для извлечения любых указанных вами данных.
- ♦ Предоставление CRUD-доступа для управления данными WordPress — это полная противоположность тому, что мы только что обсуждали. Если вы хотите, чтобы стороннее приложение могло управлять данными WordPress, то вам нужно предусмотреть соответствующую процедуру авторизации. REST API предлагает несколько разных способов аутентификации для создания, чтения, обновления и удаления данных от имени пользователя WordPress. После прохождения надлежащей аутентификации вы сможете управлять всеми данными вашего WordPress-приложения извне. Существующие сейчас методы аутентификации будут рассмотрены далее в этой главе.
- ♦ Обмен данными между сайтами на базе WordPress — это довольно распространенный вариант применения; представьте, какие возможности вам дает легкий обмен данными между двумя или более WordPress-сайтами.

- ◆ Обезглавленные сайты WordPress — "обезглавленными" называются сайты, которые работают только с серверной частью системы WordPress, не задействуя ее клиентскую часть. Иногда к подобному варианту установки WordPress прибегают исключительно с целью сохранения данных, используемых в другом приложении посредством REST API.
- ◆ Мобильные приложения на базе WordPress — именно этим целыми днями занимаются в компании AppPresser. С помощью REST API вы можете создавать мобильные приложения, использующие WordPress в качестве источника данных. Если вы создаете самостоятельное приложение, которому не нужен веб-сайт, то в качестве основы для него можно выбрать "обезглавленный" вариант установки WordPress. Если же у вас уже есть некоторый сайт, то вы можете создать мобильное приложение, которое будет служить дополнением к этому сайту и пользоваться его данными. Создание мобильных приложений на базе WordPress будет подробно рассмотрено в *главе 16*.
- ◆ Асинхронное обновление данных. Разработчики часто сталкиваются с проблемами, когда им нужно создать новый WordPress-сайт на основе моментальных снимков эксплуатируемых баз данных системы WordPress (которые обычно создаются с сохранением всего исходного контента для того, чтобы разработчики могли использовать реальные данные при создании сайта). За время проведения разработки на эксплуатируемом сайте может быть создан новый контент — посты, комментарии, аккаунты пользователей и т.д. Это может привести к серьезной проблеме, когда подходит время запуска нового сайта. В зависимости от того, что представляет собой ваш проект, вы можете "заморозить" контент на эксплуатируемом сайте, импортируя все новые данные на разрабатываемый сайт, который затем становится эксплуатируемым сайтом. С другой стороны, иногда лучше переносить все новые данные разрабатываемой базы данных (настройки, посты нового типа, метаданные и т.д.) на эксплуатируемый сайт. Если вам уже не раз приходилось запускать новый WordPress-сайт на основе существующего WordPress-сайта, то вы не понаслышке знаете, что мы имеем в виду. Когда вы уже готовы запустить новый сайт, обычно требуется обеспечить присутствие определенных данных и на эксплуатируемом, и на разрабатываемом сайте. Для этого приходится писать кастомные скрипты, кропотливо отбирающие нужные данные из обоих окружений. Хм, а не лучше ли с помощью REST API обновлять на разрабатываемом сайте любые новые или старые типы данных в режиме реального времени по мере обновления эксплуатируемого сайта? Такое решение избавит вас от необходимости тратить массу времени на ручную синхронизацию данных перед запуском нового сайта.
- ◆ Виртуальная реальность и игры — представьте себе иммерсивную среду виртуальной реальности с кастомным контентом, передаваемым посредством REST API. Например, если в гоночной игре кастомный контент может отображаться на стоящих у дороги рекламных щитах.
- ◆ Alexa и Google Home — "Alexa, сделай мне бутерброд!" Мы уверены, что это можно реализовать, используя REST API. ;)

Использование WordPress REST API версии 2

Теперь, когда вы уже знаете, для каких целей может применяться REST API, и что вообще скрывается за словом "API", давайте поговорим об использовании REST API более предметно.

Обнаружение

В последних версиях WordPress интерфейс REST API активируется по умолчанию, но вам все равно потребуется убедиться в том, что он активирован, а также узнать, к какому контенту вы имеете доступ. Самый простой способ узнать, с чем вы можете работать, состоит в том, чтобы добавить окончание `/wp-json/` или `/?rest_route=/' к любому доменному имени системы WordPress в адресной строке браузера или приложения Postman, например, так: localhost/wordpress/wp-json/.`

Ответом будет схема в формате JSON, возвращаемая всеми зарегистрированными на сайте конечными точками API.

Аутентификация

Аутентификация играет важную роль, когда использование REST API не ограничивается простым извлечением общедоступного контента из WordPress-сайта. REST API позволяет приенить различные способы аутентификации в зависимости от того, что вы создаете.

Аутентификация с помощью cookie-файлов

Аутентификация с помощью cookie-файлов задана по умолчанию, и на момент написания книги была единственным методом аутентификации, встроенным в ядро WordPress. Вы также можете использовать другие методы аутентификации с помощью сторонних плагинов или создать собственный метод аутентификации, если у вас возникнет такое желание или потребность.

Для аутентификации с помощью cookie-файлов необходимо лишь, чтобы пользователь авторизовался в WordPress. При выполнении любых запросов к API будет учитываться то, какие действия может выполнять этот авторизованный пользователь.

Для защиты от CSRF-атак REST API также задействует одноразовые коды. Поэтому, если кто-то захочет произвести вредоносные действия на вашем сайте, ему потребуется раздобыть не только ваш cookie-файл, но и правильный одноразовый код. При использовании REST API для кастомных тем и плагинов рекомендуется встроенный JavaScript API, который обеспечивает автоматическое создание одноразовых кодов. Конечно, вы также можете выполнять запросы и вручную, но при этом вам потребуется передавать одноразовый код с каждым выполняемым вами запросом.

¹ Окончание `/?rest_route=/' следует использовать для WordPress-сайтов, на которых нет постоянных ссылок.`

Аутентификация с помощью cookie-файлов позволяет вам очень легко встраивать интересные элементы и функции в клиентскую часть сайта с помощью тем или плагинов так, чтобы пользователи могли взаимодействовать с данными WordPress, не обращаясь к серверной части.

Вот пример того, как встроенный JavaScript-клиент может создавать одноразовый код:

```
wp_localize_script('wp-api', 'apiSettings', array(
    'root' => esc_url_raw(rest_url()),
    'nonce' => wp_create_nonce('wp_rest')
));
```

Этот код помещает в HTML-заголовок JavaScript-объект, который может быть использован в Ajax-запросах к конечной точке API. Передайте этот одноразовый код со своим Ajax-запросом, и он будет служить для аутентификации ваших вызовов API, а также для защиты вашего сайта от CSRF-атак. Одноразовые коды и связанные с ними проблемы безопасности были подробно рассмотрены в *главе 8*.

Далее приведен пример осуществления Ajax-вызова с использованием библиотеки jQuery.

Пример

```
$.ajax({
    url: apiSettings.root + 'wp/v2/posts/1',
    method: 'POST',
    beforeSend: function (xhr) {
        xhr.setRequestHeader('X-WP-Nonce', apiSettings.nonce);
    },
    data: {
        'title' : 'Hello World'
    }
}).done(function (response) {
    console.log(response);
});
```

Этот код обновляет заголовок поста с идентификатором 1. Одноразовый код передается в заголовке запроса X-WP-Nonce. Если вы создадите кастомные конечные точки, то проверять правильность этого одноразового кода не потребуется, так как API выполнит такую проверку автоматически.

Базовая аутентификация

Базовая аутентификация представляет собой "дешевое и сердитое" решение для разработки приложений для внешних клиентов, интегрируемых с данными WordPress. Прежде всего, следует отметить, что этот метод аутентификации не стоит использовать в эксплуатационном окружении или для тех WordPress-сайтов, которые вы хотите обезопасить от взлома (рекомендуется так относиться ко всем сай-

там). Базовая аутентификация подходит исключительно для целей разработки, поскольку она подразумевает отправку логина и пароля в формате base64 в заголовке каждого запроса к REST API. Поскольку строки в формате base64 можно перехватить и декодировать, то надо надеяться, вы понимаете, к каким проблемам это может привести.

Чтобы быстро протестировать работу REST API при создании внешнего по отношению к WordPress приложения, можно воспользоваться плагином WP-API Basic-Auth (github.com/WP-API/Basic-Auth).

Базовая аутентификация станет более безопасной, если и ваш сайт, и конечная точка API, к которой вы обращаетесь, будут настроены на обработку запросов *только* по протоколу HTTPS. С каждым запросом будет все равно передаваться одна и та же зашифрованная строка, но ее окажется сложно взломать, если вы создадите достаточно надежный пароль (см. главу 8).

Вы также можете модифицировать код базовой аутентификации так, чтобы она осуществлялась только для некоторых конкретных конечных точек. Это означает, что вы можете задать базовую аутентификацию для одной из конечных точек, не используя ее для всего WordPress-сайта. Такой способ базовой аутентификации, к примеру, характерен для плагина WP SSO, о котором мы поговорим далее в этой главе. Однако, несмотря на все сказанное, мы рекомендуем вам всегда рассматривать возможность выбора других методов аутентификации перед тем, как применять базовую аутентификацию для API своего приложения.

Вот пример запроса к REST API, выполняемого с использованием базовой аутентификации.

Пример

```
$.ajax({
  url: 'localhost/wp-json/wp/v2/posts/1',
  method: 'POST',
  beforeSend: function (xhr) {
    xhr.setRequestHeader('Authorization',
                        'Basic ' + btoa(username + ':' + password));
  },
  data:{
    'title' : 'Hello World'
  }
}).done(function (response) {
  console.log(response);
});
```

JWT-токены

JWT (JSON Web Token, веб-токен в формате JSON) — это основанный на формате JSON открытый стандарт для создания токенов доступа, предназначенных, помимо прочего, для аутентификации доступа API к веб-сервису. Это безопасный способ

обмена данными между веб-сервисом и клиентским приложением. JWT-токен представляет собой объект JSON, состоящий из заголовка, полезной нагрузки и подписи. При регистрации пользователя в сервисе аутентификации (например, при входе в каталог wp-admin) создается JWT-токен, который может использоваться им в стороннем приложении. При получении JWT-токена от стороннего приложения веб-сервис (например, REST API) проверяет его валидность, и, если с этим все нормально, разрешает клиенту работать с API для выполнения тех CRUD-действий, к которым имеет доступ соответствующий пользователь.

На наш взгляд, самый быстрый и простой способ начать безопасное использование REST API состоит в том, чтобы осуществить JWT-аутентификацию с помощью плагина WP REST API ([oreil.ly/ 9j29R](https://oreil.ly/9j29R)). Этот плагин расширяет REST API для аутентификации доступа к API посредством JWT-токенов. Настройка плагина не должна вызывать у вас затруднений; попробуйте его в действии, если вам нужен самый быстрый способ безопасного использования REST API.

OAuth-аутентификация

OAuth (Open Authorization) — это выпущенный в 2007 году открытый стандарт протокола аутентификации и авторизации с использованием токенов. Протокол OAuth — предпочтительный вариант в случае разработки приложений для внешних клиентов, интегрируемых с данными WordPress. Протокол OAuth, по сути, выступает в качестве посредника между пользователем и системой WordPress. Он позволяет выполнять аутентификацию пользователя без раскрытия его пароля в запросах. *OAuth-нотомом* называется процесс получения токена, для дальнейшей авторизации и использования в приложении конкретных данных из учетной записи пользователя.

На данный момент существуют две версии протокола OAuth: 1.0 и 2.0. Основное различие между ними состоит в том, что версия 2.0 подразумевает авторизацию запросов к API по протоколу HTTPS или SSL/TLS. Команда разработчиков REST API также создала два плагина, которые позволяют использовать в WordPress обе версии протокола OAuth.

- ◆ <https://github.com/WP-API/OAuth1> — этот плагин на самом деле основан на версии OAuth 1.0a, которая, в отличие от версии OAuth 1.0, не требует протокола SSL для каких-либо конечных точек. Поскольку WordPress не требует, чтобы вы использовали протокол SSL, или, чтобы обращение к вашему сайту осуществлялось по протоколу HTTPS, этот плагин лучше развивается и более широко распространен на данный момент.
- ◆ <https://github.com/WP-API/OAuth2> — в некоторых случаях можно рекомендовать и этот плагин, однако обратите внимание, что он еще находится на стадии бета-версии.

Конечно, вы можете создать "с нуля" собственный плагин, или воспользоваться одной из имеющихся библиотек, но, скажите, зачем вам тратить массу времени на попытки заново изобрести колесо? В обоих упомянутых плагинах протокол OAuth реализуется с помощью так называемой *"трехногой аутентификации"*,

каждая "нога" в которой представляет отдельную роль участника процесса аутентификации:

- ◆ Клиент — стороннее приложение, для которого нужно обеспечить взаимодействие с WordPress.
- ◆ Сервер — установленная система WordPress, к которой будет выполняться API-запросы стороннее приложение.
- ◆ Владелец ресурса — конечный пользователь, который располагает логином для входа в установленную систему WordPress и использует стороннее приложение.

Протокол OAuth использует токены доступа, которые выдаются *сервером*, когда *владелец ресурса* проходит аутентификацию, введя свои учетные данные. Благодаря токенам *клиент* получает доступ к серверу. Сервер может в любой момент отозвать токены доступа, выданные определенному владельцу ресурса. Кроме того, срок действия этих токенов ограничен, и при его истечении владельцу ресурса требуется проходить аутентификацию заново.

Далее представлено подробное описание OAuth-потока:

- ◆ Клиент отправляет на сервер подписанный запрос для получения токенов запроса, или, иначе говоря, временных токенов доступа. Этот запрос отправляется на URI-адрес конечной точки временных токенов доступа и содержит следующие данные:
 - `oauth_consumer_key` — ключ, предоставляемый сервером;
 - `oauth_timestamp`;
 - `oauth_nonce`;
 - `oauth_signature`;
 - `oauth_signature_method`;
 - `oauth_callback`;
 - `oauth_version` — опциональный параметр.
- ◆ Сервер проверяет валидность запроса, и если все нормально, предоставляет токен запроса, содержащий следующие данные:
 - `oauth_token`;
 - `oauth_token_secret`;
 - `oauth_callback_confirmed`.
- ◆ Затем клиент перенаправляет владельца ресурса (пользователя) на сервер для авторизации запроса. Для этого составляется URI-адрес запроса путем добавления токена `oauth_token` (полученного на предыдущей стадии) к URI-адресу конечной точки авторизации владельца ресурсов.
- ◆ Владелец ресурса (пользователь) авторизуется на сервере путем предоставления учетных данных.

- ◆ Если на первом шаге был представлен URI-адрес `oauth_callback`, сервер перенаправляет клиента на этот URI-адрес, добавляя следующие данные в виде строк запроса:
 - `oauth_token` — токен, полученный на втором шаге;
 - `oauth_verifier` — идентификатор, гарантирующий, что к клиенту будет перенаправлен тот же владелец ресурса, которому был предоставлен доступ.
- ◆ Если на первом шаге не был представлен URI-адрес `oauth_callback`, то сервер отображает значение параметра `oauth_verifier`, чтобы владелец ресурса мог проинформировать клиента вручную.
- ◆ После получения параметра `oauth_verifier` клиент запрашивает у сервера токены доступа путем отправки запроса на URI-адрес конечной точки запроса токена. Этот запрос содержит следующие данные:
 - `oauth_token` — токен, полученный на втором шаге;
 - `oauth_verifier` — идентификатор, полученный на предыдущем шаге;
 - `oauth_consumer_key` — ключ, предоставляемый поставщиком ресурса (сервером) перед запуском OAuth-квентирования;
 - `oauth_signature`;
 - `oauth_signature_method`;
 - `oauth_nonce`;
 - `oauth_version`.
- ◆ Сервер проверяет запрос и предоставляет следующие токены доступа:
 - `oauth_token`;
 - `oauth_token_secret`.
- ◆ Затем клиент использует предоставленные токены доступа для доступа к нужным ему данным на сервере.

Если вам еще не приходилось иметь дело с протоколом OAuth, то, взглянув на это описание процесса, вы можете подумать, что настройка этого протокола потребует больших усилий. Однако благодаря плагину для WordPress, созданному командой разработчиков REST API, этот процесс сводится лишь к нескольким простым шагам. Так что можете не волноваться, если вы еще не разобрались во всех тонкостях OAuth-потока. При необходимости вы всегда сможете разобраться в деталях этого процесса, ознакомившись с внутренним содержимым этого плагина. Вот она, красота открытого исходного кода!

Вы можете скачать плагин OAuth1 непосредственно из репозитория плагинов для WordPress (wordpress.org/plugins/rest-api-oauth1/). Попробуйте применить этот плагин на разрабатываемом вами сайте. Для этого установите и активируйте его, как любой другой плагин для WordPress.

Маршруты и конечные точки

В *главе 9* мы узнали, как можно получить доступ к нашим Ajax-сервисам через URL-адрес `/wp-admin/admin-ajax.php`. При использовании REST API каждый метод API обладает собственным URL-адресом, включающим в себя *маршрут* и *конечную точку*.

Что такое маршрут?

Маршрут — это HTTP-адрес ресурса. Так, например, маршрут постов может выглядеть следующим образом: `/wp-json/wp/v2/posts`. Вы можете использовать конечные точки для выполнения CRUD-операций, если они определены для маршрута.

Что такое конечная точка?

Конечная точка вызывает или запускает определенный метод или функцию при обращении к конкретному маршруту. У каждого маршрута может быть несколько разных конечных точек.

Вот как выглядит обращение к маршруту при наличии нескольких конечных точек:

- ◆ `GET /wp/post/1` — этот маршрут может иметь метод или функцию `get_post()` для извлечения указанного поста.
- ◆ `POST /wp/posts/1` — этот маршрут может иметь метод или функцию `update_post()` для обновления указанного поста.

Хотя мы переходим по одному и тому же маршруту `/wp/post/1`, CRUD-операция указывает, какой метод или функцию следует запустить; эта функция и является конечной точкой.

Что такое пространство имен?

Пространство имен — это "префикс" маршрутов API, используемый для идентификации конечных точек и исключения конфликта конечных точек.

Базовое пространство имен — `wp`. Оно является зарезервированным, в силу чего его не следует использовать для создания кастомных конечных точек. Если после того, как вы зарегистрируете маршрут, другой плагин или тема зарегистрирует в точности такой же маршрут, то первоначальный маршрут станет недействительным.

Запросы

Когда вы запрашиваете конечную точку API, API использует класс `WP_REST_Request` — базовый класс для реализации объекта REST-запроса.

Этот объект запроса содержит много полезных методов для обработки запроса. Так используя метод `get_posts()` для строки `/wp/posts?hello=world`, следующей после списка параметров, вы перейдете по соответствующему адресу. Метод `get_params()` в данном случае извлечет массив, содержащий параметр `hello` со значением `world`.

Также доступны следующие методы:

- ◆ `add_header` — добавляет значение для заданного заголовка;
- ◆ `canonicalize_header_name` — обеспечивает представление имен заголовков в стандартном формате;
- ◆ `from_url` — извлекает объект `WP_REST_Request` из полного URL-адреса;
- ◆ `get_attributes` — извлекает атрибуты запроса;
- ◆ `get_body` — извлекает содержимое тела запроса;
- ◆ `get_body_params` — извлекает параметры из тела;
- ◆ `get_content_type` — извлекает тип содержимого запроса;
- ◆ `get_default_params` — извлекает параметры по умолчанию;
- ◆ `get_file_params` — извлекает параметры составного файла из тела;
- ◆ `get_header` — извлекает заданный заголовок из запроса;
- ◆ `get_header_as_array` — извлекает значения заголовков из запроса;
- ◆ `get_headers` — извлекает все заголовки из запроса;
- ◆ `get_json_params` — извлекает параметры из тела, представленного в формате JSON;
- ◆ `get_method` — извлекает метод HTTP запроса;
- ◆ `get_param` — извлекает параметр из запроса;
- ◆ `get_parameter_order` — извлекает порядок приоритета параметров;
- ◆ `get_params` — извлекает в объединенном виде параметры из запроса;
- ◆ `get_query_params` — извлекает параметры из строки запроса;
- ◆ `get_route` — извлекает маршрут, соответствующий запросу;
- ◆ `get_url_params` — извлекает параметры из маршрута;
- ◆ `has_valid_params` — определяет по атрибутам запроса, является ли он валидным;
- ◆ `offsetExists` — проверяет, задан ли параметр;
- ◆ `offsetGet` — извлекает параметр из запроса;
- ◆ `offsetSet` — задает параметр запроса;
- ◆ `offsetUnset` — удаляет параметр из запроса;
- ◆ `parse_body_params` — выполняет синтаксический разбор параметров в теле запроса;
- ◆ `parse_json_params` — выполняет синтаксический разбор параметров, представленных в формате JSON;
- ◆ `remove_header` — удаляет все значения заголовка;
- ◆ `sanitize_params` — санирует (если это возможно) параметры запроса;
- ◆ `set_attributes` — задает атрибуты запроса;

- ◆ `set_body` — задает содержимое тела;
- ◆ `set_body_params` — задает параметры в теле;
- ◆ `set_default_params` — задает параметры по умолчанию;
- ◆ `set_file_params` — задает параметры составного файла в теле;
- ◆ `set_header` — задает заголовок запроса;
- ◆ `set_headers` — задает заголовки запроса;
- ◆ `set_method` — задает метод HTTP запроса;
- ◆ `set_param` — задает параметр запроса;
- ◆ `set_query_params` — задает параметры в строке запроса;
- ◆ `set_route` — задает маршрут, соответствующий запросу;
- ◆ `set_url_params` — задает параметры в маршруте.

Ответы

"Близким родственником" класса `WP_REST_Request` является класс `WP_REST_Response` — базовый класс для реализации объекта REST-ответа.

Этот класс используется для подготовки данных ответа, кода состояния HTTP и имеющихся заголовков ответа:

- ◆ `add_link` — добавляет ссылку в ответ;
- ◆ `add_links` — добавляет несколько ссылок в ответ;
- ◆ `as_error` — извлекает объект `WP_Error` из ответа;
- ◆ `get_curies` — извлекает CURIE-адреса (компактные URI-адреса), используемые для отношений;
- ◆ `get_links` — извлекает ссылки ответа;
- ◆ `get_matched_handler` — извлекает обработчик, который использовался для получения ответа;
- ◆ `get_matched_route` — извлекает маршрут, который использовался для получения ответа;
- ◆ `is_error` — проверяет, является ли ответ сообщением об ошибке, то есть, выполняется ли равенство: код ответа ≥ 400 ;
- ◆ `link_header` — задает заголовок ссылки;
- ◆ `remove_link` — удаляет ссылку из ответа;
- ◆ `set_matched_handler` — задает обработчик, который использовался для получения ответа;
- ◆ `set_matched_route` — задает маршрут (регулярное выражение для пути), который использовался для получения ответа.

Добавление собственных маршрутов и конечных точек

Какие бы мощные возможности вам ни давали встроенные маршруты и конечные точки REST API, настоящее веселье начинается тогда, когда вы начинаете создавать собственные маршруты и конечные точки. Новые конечные точки позволяют предоставлять кастомные типы постов и другие кастомные данные, имеющиеся в вашем приложении, сторонним сервисам или JavaScript-коду, запускаемому в клиентской части вашего приложения.

Во время написания этой книги мы создали плагин WP Single Sign-On (github.com/strangerstudios/wp-sso), чтобы продемонстрировать некоторые возможности REST API, включая добавление собственных маршрутов и конечных точек. Слова "Single Sign-On" (SSO) переводятся как "единая регистрация", под чем понимается возможность доступа к нескольким отдельным сайтам с использованием одних и тех же учетных данных. Существует много методов выполнения единой регистрации (SSO) как таковой, а также много других плагинов и сторонних сервисов для выполнения ее в WordPress. Свое решение мы постарались сделать простым и немудреным, чтобы его можно было легко взять в качестве образца. Этот плагин показывает, как можно создать и использовать кастомный маршрут REST API.

Функция

register_rest_route(\$namespace, \$route, \$args, \$override)

Функция `register_rest_route()` предназначена для регистрации новых маршрутов REST API. Ее следует вызывать при срабатывании хука действия `rest_api_init`.

- ◆ `$namespace` — сегмент URL-адреса, расположенный между префиксом ядра² и добавляемым вами маршрутом. Эта строка должна быть специфичной для вашего плагина или приложения.
- ◆ `$route` — строка базового URL-адреса добавляемого вами маршрута.
- ◆ `$args` — массив параметров конечной точки. Также можно передать массив массивов при снабжении конечной точки несколькими методами. Наиболее часто встречаются следующие аргументы:
 - `methods` — для определения доступных для конечной точки HTTP-методов;
 - `callback` — для определения функции обратного вызова для обработки запросов к конечной точке;
 - `args` — для определения параметров запроса, передаваемых в конечную точку.
- ◆ `$override` — параметр, указывающий, должен ли новый маршрут переопределять существующие маршруты с таким же именем? Если данный параметр будет

² По умолчанию задан префикс `wp-json`.

равен `true`, то новый маршрут будет переопределять существующие маршруты. Если данный параметр будет равен `false`, то маршруты будут объединяться, при этом более новые параметры будут иметь приоритет. Значение по умолчанию — `false`.

Чтобы посмотреть, как можно добавить собственный маршрут, давайте ознакомимся с содержимым уже упоминавшегося плагина WP Single Sign-On. Наряду с кодом для настройки конечной точки REST API и обработки запросов к ней, данный плагин также содержит код для выполнения вызовов этой конечной точки REST API в PHP-коде.

Настройка плагина WordPress Single Sign-On

Чтобы опробовать плагин WP Single Sign-On в действии, скачайте и активируйте его на двух отдельных WordPress-сайтах. На обоих сайтах выберите пункт меню **Settings** → **WP SSO** (Настройки → WP SSO). Задайте одному сайту роль *хоста*, а другому — роль *клиента*. После этого нужно скопировать содержимое поля *Host URL* (URL-адрес хоста) на сайте-хосте и вставить его в настройках для сайта-клиента. Этот URL-адрес хоста представляет собой URL-адрес кастомной конечной точки REST API, которую создал плагин.

При активации плагина на обоих сайтах вы сможете авторизоваться на сайте-клиенте с помощью логина и пароля действительного пользователя сайта-хоста. При этом пользователи сайта-клиента (например, администраторы) смогут по-прежнему авторизоваться с помощью имеющихся у них логинов и паролей для этого сайта. Плагин сверяется только с сайтом-хостом, если логин по какой-либо причине дает сбой на сайте клиента.

Добавление маршрута `/wp-sso/v1/check`

Сначала необходимо зарегистрировать наш новый маршрут. Мы подключаемся к хуку действия `rest_api_init` и определяем маршрут с помощью функции `register_rest_route()` как показано в следующем примере.

Пример

```
function wpsso_register_routes() {
    $options = wpsso_get_options();

    //Проверяем, активирован ли параметр host
    if (! $options['host']) {
        return;
    }

    register_rest_route(
        'wp-sso/v1',
        '/check',
```

```

        array(
            'methods' => WP_REST_Server::READABLE,
            'callback' => 'wpsso_check_authentication_endpoint',
        )
    );
}

add_action('rest_api_init', 'wpsso_register_routes');

```

Первым делом эта функция принимает массив параметров для плагина WordPress Single Sign-On. Если параметр `host` не активирован, то данный маршрут не регистрируется, поскольку его задача состоит в том, чтобы сделать сайт хостом.

Мы указали здесь пространство имен `wp-sso/v1`, т. е. дополнили слаг плагина еще одним сегментом URL-адреса — `v1`. Если впоследствии нам потребуется внести в наш API критически важные изменения, можно будет сохранить старый API как версию `v1` и создать новые маршруты, задав префикс `v2`. Это позволит пользователям нашего API делать явный выбор в пользу новой версии. Мы добавляем конечную точку `check`, которая будет настроена на проверку того, работают ли логин и пароль, переданные посредством базовой аутентификации, для действительного пользователя на сайте-хосте.

Передаваемые нами аргументы определяют доступные методы и функцию обратного вызова для обработки запросов к маршруту. Значение `WP_REST_Server::READABLE` является псевдонимом для метода `GET`. Следующий пример иллюстрирует, как выглядит функция обратного вызова `wpsso_check_authentication_endpoint()`.

Пример

```

function wpsso_check_authentication_endpoint() {
    global $current_user;

    if (! empty($current_user->user_login)) {
        $r = array(
            'success' => true,
            'message' => sprintf('Logged in as %s', $current_user->user_login),
            'user_login' => $current_user->user_login,
            'user_email' => $current_user->user_email,
            'first_name' => $current_user->first_name,
            'last_name' => $current_user->last_name,
        );
    } else {
        $r = array(
            'success' => false,
            'message' => 'Not logged in.',
            'user_email' => null,
            'user_login' => $current_user->user_login,
            'first_name' => null,

```

```

        'last_name' => null,
    );
}

$r = rest_ensure_response($r);

return $r;
}

```

Данная функция обратного вызова проверяет, авторизовался ли пользователь. Если пользователь авторизовался, мы присваиваем переменной `success` значение `true` и передаем вместе с ней логин, адрес электронной почты, имя и фамилию пользователя. Если пользователь не авторизовался, то переменная `success` получает значение `false`, а переменные для данных о пользователе заполняются нулями.

Обратите внимание, что возвращаемый массив в этом примере передается с помощью функции `rest_ensure_response()`. Эта функция преобразует массив в объект `WP_REST_Response`. Как вы увидите далее, при выполнении запросов к этой новой конечной точке будет возвращаться JSON-объект, содержащий массив значений, определенных нами в функции обратного вызова.

Подключение к нашему плагину базовой аутентификации

Чтобы работали наши проверки переменной `$current_user`, эти запросы к API должны проходить аутентификацию. Для этого мы подключили класс базовой аутентификации, определенный в файле `/includes/basic-auth.php`. Это такой же обработчик аутентификации, как тот обработчик, ссылка на который дается в руководстве по WordPress REST API. Мы лишь снабдили имена функций префиксом во избежание конфликтов и добавили код, приведенный в следующем примере, чтобы базовая аутентификация активировалась только для HTTPS-трафика, поступающего на нашу конкретную конечную точку.

Пример

```

//Не выполняем, если не используется SSL
if (! is_ssl()) {
    return $user;
}

//не выполняем, если не используется наш маршрут
if (! empty($_REQUEST['rest_route'])) {
    $rest_route = '/' . rest_get_url_prefix() . $_REQUEST['rest_route'];
} else {
    $rest_route = $_SERVER['REQUEST_URI'];
}

```

```
,
if ($rest_route != '/' . rest_get_url_prefix() . '/wp-sso/v1/check') {
    return $user;
}
```

Использование настроенной нами конечной точки для проверки учетных данных пользователя

WordPress-сайты, на которых будет запущен наш плагин Single Sign-On с настройкой сайта в качестве хоста, будут иметь описанную выше конечную точку REST API, при доступе к которой аутентифицированного пользователя будет возвращаться массив с подробной информацией об этом пользователе. Сайты-клиенты будут обращаться к этой конечной точке API во время прохождения пользователями авторизации, чтобы проверять, существует ли пытающийся авторизоваться пользователь на сайте-хосте. Давайте посмотрим, как выглядит этот код.

В основном файле плагина wp-sso.php определена функция `wpsso_authenticate()` для подключения к хуку действия `wp_authenticate`. Убедившись в том, что логин и пароль были введены, эта функция проверяет, работают ли эти учетные данные на локальном сайте. Если они не срабатывают на локальном сайте, функция пробует использовать их в запросе базовой аутентификации к конечной точке плагина WordPress Single Sign-On на сайте-хосте. Следующий пример показывает, как выглядит этот запрос.

Пример

```
$url = $options['host_url'];
$args = array(
    'headers' => array(
        'Authorization' => 'Basic ' . base64_encode($username . ':' . $password),
    ),
);
$response = wp_remote_get($url, $args);
```

URL-адрес хоста мы берем из массива параметров. Мы также определяем массив заголовков, передаваемых с запросом GET. Мы кодируем переменные `$username` и `$password`, которые были переданы хуку `wp_authenticate`, и включаем их в заголовок авторизации. Поскольку мы активировали базовую аутентификацию для нашей конечной точки на сайте-хосте, сайт-хост попытается аутентифицировать запрос, используя присланные логин и пароль. Мы выполняем запрос с помощью функции `wp_remote_get()`.

Если эти учетные данные не сработают на сайте-хосте, то в ответ на запрос будет возвращен код ошибки 500 с сообщением "неправильный пароль".

Если же эти учетные данные сработают на сайте-хосте, ответ будет выглядеть как показано в следующем примере.

Пример

```
array(6) {
  ["headers"]=>
    ...
}
["body"]=>
string(162) '{"success":true,...}'
["response"]=>
array(2) {
  ["code"]=>
    int(200)
  ["message"]=>
    string(2) "OK"
}
["cookies"]=>
array(0) {
}
["filename"]=>
NULL
["http_response"]=>
...
}
```

В оставшейся части функции `wpsso_authenticate()` осуществляется проверка, равен ли 200 код состояния ответа, и если это так, из тела ответа извлекаются данные о пользователе с помощью функции `json_decode()`. Если на локальном сайте существует пользователь с таким же адресом электронной почты, его пароль обновляется. Если на локальном сайте нет пользователя с таким же адресом электронной почты, создается новая учетная запись пользователя.

Популярные плагины, использующие WordPress REST API

Многие плагины посредством REST API не только поддерживают пользовательский интерфейс, но и предоставляют доступ к данным и функциональности для других приложений. Далее перечислены некоторые из тех плагинов, которые работают с WP REST API особенно хорошо и могут быть рекомендованы в качестве образца.

WooCommerce

Плагин WooCommerce обладает полнофункциональным API, который позволяет выполнять почти все, что требуется делать в этом плагине, за исключением обработки оплаты. Вы можете выполнять CRUD-операции над продуктами, заказами, купонами и покупателями. Вы можете получать доступ к отчетам и информации о налоговых ставках. Мы настоятельно рекомендуем вам ознакомиться с подробной технической документацией по WooCommerce REST API (oreil.ly/_WZ0f).

WooCommerce REST API реализует базовую аутентификацию на основе протокола SSL. Однако при этом не используются те же логин и пароль, с помощью которых осуществляется вход в панель администратора WordPress. Вместо этого в настройках плагина WooCommerce имеется страница для генерирования наборов API-ключей для конкретных пользователей. Этим API-ключам можно назначить разрешения на чтение, запись или чтение/запись.

Хотя эти ключи обозначены как "ключ покупателя" и "секретный код покупателя", в API-запросах базовой аутентификации они играют роль, соответственно, логина и пароля.

Пример: скрывание продающих баннеров для платящих покупателей

WooCommerce API возвращает объект `Customer`, у которого есть полезное свойство `is_paying_customer`. В некоторых случаях требуется, чтобы определенные баннеры или реклама отображались только тем пользователям, которые еще не успели стать вашими покупателями. Далее приведен небольшой пример того, как это можно реализовать в JavaScript-коде с помощью WooCommerce API.

Прежде всего, нам нужен PHP-код, который будет регистрировать, локализовать и подключать наш JavaScript-код. Этот фрагмент кода аналогичен показанному ранее в этой главе примеру аутентификации с помощью cookie-файлов, но теперь мы также передаем идентификатор текущего пользователя и подключаем наш JavaScript-код.

Пример

```
function my_hide_sale_banner_script() {
    global $current_user;

    wp_register_script(
        'hide-sale-banner',
        plugins_url('js/hide-sale-banner.js', __FILE__),
        array('jquery')
    );

    wp_localize_script('hide-sale-banner', 'HSBSSettings', array(
        'root' => esc_url_raw(rest_url()),
    ));
}
```

```

        'nonce' => wp_create_nonce('wp_rest'),
        'current_user_id' => $current_user->ID,
    ));

    wp_enqueue_script('hide-sale-banner');
}
add_action('wp_enqueue_scripts', 'my_hide_sale_banner_script');

```

Приведенный в следующем примере JavaScript-код получает информацию о текущем пользователе с помощью WooCommerce API и, если пользователь уже является платящим покупателем, удаляет со страницы элемент баннера.

Пример

```

jQuery(document).ready(function() {
    jQuery.ajax({
        url: HSBSettings.root + 'wc/v3/customers/'
            + HSBSettings.current_user_id,
        method: 'GET',
        beforeSend: function (xhr) {
            xhr.setRequestHeader('X-WP-Nonce', HSBSettings.nonce);
        },
    }).done(function (customer) {
        if (customer['is_paying_customer']) {
            jQuery('#sale-banner').remove();
        }
    });
});

```

Вот и все! WooCommerce API также может использоваться для оперативного генерирования купонов, обновления баннера "Обслужено X покупателей!" на основе статистических данных о продажах, оперативного обновления информации о продукте при изменении его описания в других системах и т.д.

BuddyPress

Разработчики плагина BuddyPress ведут работу по реализации CRUD-операций интерфейса REST API для многочисленных типов данных этого плагина. На момент написания книги было выполнено уже примерно 90% этой работы³.

Работа над BuddyPress REST API ведется в рамках разработки плагина, расширяющего возможности плагина BuddyPress (github.com/buddypress/BP-REST). Для использования данного API необходимо установить и активировать этот плагин в

³ Конечно, стоит учесть, что на выполнение последних 10% любого проекта обычно уходит не меньше времени, чем на выполнение предыдущих 90%.

дополнение к плагину BuddyPress. В дальнейшем планируется встроить поддержку REST API в версию 5.0 плагина BuddyPress.

Хотя BuddyPress REST API еще находится в стадии активной разработки, уже сейчас он предлагает ряд потрясающих возможностей. Уже имеются конечные точки для активностей, групп, полей и групп профилей, членов, уведомлений, компонентов и многого другого.

Для большинства этих конечных точек требуется, чтобы аутентификация возвращала данные (например, путем использования WordPress REST API с помощью плагина OAuth 1.0a Server (oreil.ly/-WR6o)).

Пример: выделение активностей конкретных пользователей

По умолчанию конечная точка `/wp-json/buddypress/v1/activity` возвращает последний элемент активности для всех пользователей. Вы можете сузить область поиска, передав в качестве параметра идентификатор конкретного пользователя `user_id`, а также изменить количество возвращаемых результатов с помощью параметра `per_page`. Так, например, при поступлении аутентифицированного запроса на URL-адрес вида `https://yoursite.com/wp-json/buddypress/v1/activity?user_id=1&per_page=1` будет возвращен последний элемент активности пользователя с идентификатором 1.

После этого можно с помощью приведенного далее примера кода извлечь последний элемент активности этого пользователя и отобразить его более заметным образом в верхней части страницы.

Пример

```
function my_highlight_admin_activity_script() {
??
<style>
    li.activity.highlighted {
        background-color: #FFFFCC;
    }
</style>
<script>
    fetch('/wp-json/buddypress/v1/activity?user_id=1&per_page=1')
    .then(function(response) {
        return response.json();
    })
    .then(function(activity) {
        const elements = document.querySelectorAll(
            "a[href='" + activity[0].link + "']"
        );
        elements.forEach(function(element) {
            const wrapping_div = element.closest('li.activity');
            wrapping_div.classList.add('highlighted');
```



```

    });
});
</script>
<?php
}
add_action('wp_footer', 'my_highlight_admin_activity_script');

```

Если мы добавим этот JavaScript-код в плагин или файл темы `functions.php`, то он будет добавляться в подвал всех страниц клиентской части. Однако обычно такой JavaScript-код размещают в отдельном файле `*.js` и подключают его так, как это делается в данном примере; мы лишь хотели сказать, что существуют и другие способы.

В данном примере мы заменяем метод `jQuery.ajax` методом `fetch()` для получения данных от BuddyPress REST API. Мы также используем селекторные команды "ванильной" разновидности языка JavaScript вместо соответствующих конструкций `jQuery`. Этот код должен работать в большинстве современных браузеров без дополнительной загрузки `jQuery API`.

Поскольку конечная точка BuddyPress REST API для активности не требует аутентификации, нам не нужно выполнять аутентификацию для получения этих данных. Мы просто переходим в конечную точку `/wp-json/buddypress/v1/activity?user_id=1&per_page=1`, передаем параметр `user_id=1` для получения элементов активности только этого пользователя, и ограничиваем количество результатов до одного последнего элемента активности с помощью параметра `per_page=1`.

Paid Memberships Pro

Плагин Paid Memberships Pro имеет простой, но мощный REST API, который позволяет получить ответ на очень важный вопрос о том, является ли пользователь платящим членом?



Хотя данный плагин поддерживает и другие конечные точки, количество которых еще планируется расширить, особый интерес представляет только добавляемая им конечная точка `/wp-json/wp/v2/users/2/pmpo_membership_level`, которая служит для подключения других приложений к базе данных членов, сохраняемой в системе WordPress.

В отличие от упоминавшихся ранее плагинов, которые создают для своих API собственные новые маршруты, плагин Paid Memberships Pro дополняет существующий маршрут `/users/` новой конечной точкой для проверки уровня членства пользователя.

Так, например, данная конечная точка может использоваться в рецепте Zapier или IFTTT для проверки членства пользователя перед принятием решения о том, в какое хранилище CRM-системы следует отправить данные о пользователе. С помощью этой конечной точки также можно управлять платящими членами в WordPress при контроле доступа к приложению, созданному на основе любой платформы, позволяющей выполнять аутентифицированные вызовы API.

Пример: проверка адреса электронной почты на предмет членства

Для использования конечной точки `pmp_pro_membership_level` сначала нужно получить идентификатор пользователя. Это можно сделать, выполнив поиск с помощью встроенной конечной точки `users`. Затем мы передаем этот идентификатор в запрос на получение данных об уровне членства пользователя.

Обратите внимание, что в следующем примере задана базовая аутентификация. На момент написания книги плагин **Paid Memberships Pro** не предлагал какой-либо собственный метод аутентификации. Вам потребуется активировать плагин базовой аутентификации либо внести изменения в этот код.

Пример

```
function my_check_host_site_membership() {
    global $current_user;

    // Измените эти значения
    $host_site_url = 'https://hostsite.com/';
    $restricted_post_id = 2;
    $apiuser = 'apiuser';
    $apipassword = 'apipassword';

    // Мы блокируем только некоторый конкретный идентификатор поста
    $queried_object = get_queried_object();
    if (empty($queried_object)
    || $queried_object->ID != $restricted_post_id) {
        return;
    }

    // Если не выполнена авторизация, перенаправляем на сайт-хост
    if (! is_user_logged_in()) {
        wp_redirect($host_site_url);
        exit;
    }

    // Проверяем на членство на сайте-хосте
    $url = esc_url(
        $host_site_url
        . '/wp-json/wp/v2/users/?search='
        . urlencode($current_user->user_email)
    );

    $args = array(
        'headers' => array(
            'Authorization' => 'Basic '
```

```

        . base64_encode($apiuser . ':' . $apipassword),
    ),
);

// Проверяем, был ли успешным наш первый запрос
$response = wp_remote_get($url, $args);
if (empty($response) || $response['response']['code'] != '200') {
    wp_redirect($host_site_url);
    exit;
}

// Убеждаемся, что пользователь найден.
$response_body = json_decode($response['body']);
if (empty($response_body)) {
    wp_redirect($host_site_url);
    exit;
}

// Поиск пользователя возвращает массив. Берем первое значение
$host_user = $response_body[0];

// Проверяем членство пользователя на сайте-хосте
$url = esc_url(
    $host_site_url
    . '/wp-json/wp/v2/users/'
    . $host_user->id
    . '/pmpo_membership_level'
);

$response = wp_remote_get($url, $args);

// Проверяем, был ли успешным второй запрос
if (empty($response) || $response['response']['code'] != '200') {
    wp_redirect($host_site_url);
    exit;
}

// Проверяем уровень членства
$membership_level = json_decode($response['body']);
if (empty($membership_level)) {
    wp_redirect($host_site_url);
}

exit;

/*
    В этой точке переменная $membership_level будет
    содержать информацию об уровне пользователя. Мы
    можем выполнить проверку на обладание некоторым

```

конкретным уровнем, либо просто остановиться здесь,
позволив просматривать эту страницу пользователям
всех уровней.

*/

}

```
add_action('template_redirect', 'my_check_host_site_membership');
```

В этом примере необходимо поменять URL-адрес сайта-хоста на адрес главной страницы WordPress-сайта, на котором будет активирован плагин Paid Memberships Pro для управления членами. На этот URL-адрес также будут перенаправляться пользователи. Переменная `$restricted_post_id` содержит идентификатор поста, который вы хотите заблокировать для членов. Вы можете модифицировать этот пример так, чтобы блокировались некоторые метаданные поста, массив постов или какой-либо другой признак. Если на сайте, использующем плагин Paid Memberships Pro, будет активирована базовая аутентификация, вам останется лишь занести в переменные `$apiuser` и `$apipassword` логин и пароль пользователя с полномочием `edit_users`.

REST API — это мощный инструмент, дающий вам больше скорости и гибкости в плане создания функциональных возможностей и приложений поверх WordPress. Примером встроенного компонента, активно использующего REST API, может служить новый блочный редактор Gutenberg. В следующей главе мы подробно обсудим редактор Gutenberg, блоки и создание кастомных типов постов.

Проект Gutenberg, блоки и кастомные типы блоков

Когда в январе 2017 года был предложен новый редактор для WordPress, Мэтт Мулленберг написал следующее (oreil.ly/CIAYx):

"Данный редактор призван обеспечить новый опыт создания новых страниц и постов, который упрощает написание объемных постов и предлагает "блоки" для легкого создания тех вещей, которые сегодня требуют использования коротких кодов, кастомного HTML-кода или малопонятных встроенных представлений".

Не прошло и двух лет, как редактор блоков Gutenberg был включен в состав версии 5.0 системы WordPress, принеся с собой новый способ редактирования постов и разработки с помощью WordPress.



"Gutenberg" — это кодовое название исходного проекта по разработке редактора блоков. Сегодня мы предпочитаем говорить *"редактор блоков"* или просто *"редактор системы WordPress"*, однако часто используется и название "Gutenberg", под которым подразумевается сам редактор.

Чтобы помочь и пользователям, и разработчикам в освоении возможностей редактора Gutenberg, команда его создателей составила "Руководство по редактору блоков" ([oreil.ly/ R8JyX](https://oreil.ly/R8JyX)). Это компактная, хорошо написанная и постоянно дорабатываемая документация, которую обязательно следует прочитать. Мы рекомендуем вам ознакомиться с ней прямо сейчас, а затем вернуться к этому месту. В представленном далее материале будут даны ссылки на некоторые разделы этого руководства.

В данной главе мы кратко рассмотрим основные особенности редактора блоков, создадим в качестве отправной точки простейший блок, после чего чуть подробнее коснемся тех возможностей, которые представляют наибольший интерес при разработке приложений.

Редактор блоков реализован главным образом с помощью JavaScript-кода (использующего пользовательский интерфейс) и администрируется как проект Node.js. Хотя руководство по редактору Gutenberg может оказать вам реальную помощь в программировании блоков, в представленных в нем примерах очень сложно понять, к чему относится та или иная часть кода — к JavaScript-коду, фреймворку React или редактору Gutenberg. Понимание того, как взаимодействуют друг с другом эти технологии, может оказаться очень полезным при разработке расширений для WordPress. В своих примерах мы постарались как можно яснее показать, откуда берется тот или иной код.

Редактор системы WordPress

Базовым элементом нового редактора системы WordPress являются блоки, которые располагаются один за другим и представляют абзацы, заголовки, списки, изображения и более сложные компоненты. Некоторые типы блоков, например блоки группы и колонки, могут содержать вложенные блоки. Вы можете менять расположение блоков, перетаскивая их в окне редактора (рис. 11.1).

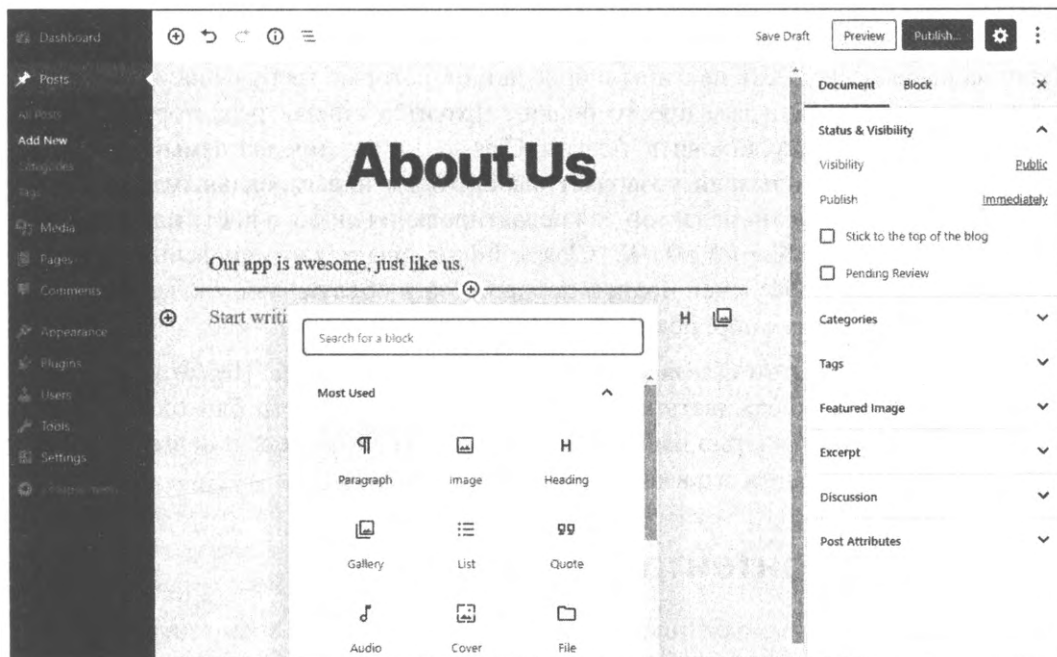


Рис. 11.1. Редактор блоков системы WordPress

Вы можете добавлять блоки, выполняя щелчок по кнопке + в пользовательском интерфейсе или вводя символ / и имя блока в пустом месте в окне редактора. Нажатие клавиши ввода внутри блока абзаца ведет к созданию нового блока абзаца. Некоторые типы блоков создаются автоматически при вводе определенных символов разметки внутри пустого блока. Так, символ * начинает маркированный список, а символы ## — вторичный заголовок¹.

Каждый блок редактируется по отдельности. Для этого нужно щелкнуть по блоку, чтобы передать ему фокус и переключиться в окно редактора. При этом вкладка **Block** (Блок) правой боковой панели будет содержать настройки, специфичные для редактируемого блока. Выполнив щелчок по другому блоку или по пустому месту в окне редактора, вы снова вернетесь к клиентской части приложения.

¹ Полный список форматизирующих кодов и полезных клавишных комбинаций можно найти на сайте WordPress (oreil.ly/0StKa).

Редактор системы WordPress будет в максимально возможной степени зеркалировать клиентскую часть. Хотя для упрощения редактирования блоки могут отожделяться с метками-заполнителями и некоторыми отличиями в форматировании, разработчикам рекомендуется делать так, чтобы внешний вид создаваемых ими блоков в окне редактора имел как можно больше сходства с их видом в клиентской части.

Плагин Classic Editor

Если на вашем сайте есть плагины или функции, которые требуют наличия классического редактора, или вам просто больше нравится старый редактор в стиле TinyMCE, вы можете установить плагин Classic Editor, предлагаемый по адресу oreil.ly/BqQ9K. Этот плагин позволяет выборочным или глобальным образом использовать классический редактор для редактирования любого поста или страницы. Как указывается в файле *README*, "Classic Editor является официальным плагином WordPress, и будет обеспечен полной поддержкой и обслуживанием, как минимум, до 2022 года или до тех пор, пока это будет необходимо".

Мы рекомендуем плагин Classic Editor лишь в крайнем случае. Несмотря на то, что его планируется поддерживать вплоть до 2022 года, редактор блоков постепенно станет неотъемлемой частью панели администратора WordPress, и новые плагины и темы будут использовать его возможности.

Блоки для контента и дизайна

Большинство предлагаемых типов блоков имеют отношение к контенту и дизайну. Они позволяют создавать насыщенные макеты дизайна, не прибегая к написанию кода. Поскольку обсуждение популярных сегодня блоков для представления разделителей, каруселей или галерей выходит за рамки данной книги, мы лишь отметим, что у вас будет достаточный выбор таких блоков.

Блоки для представления функциональности

Часть блоков служит главным образом для представления функциональности сайта или приложения на базе WordPress. Так в плагине Paid Memberships Pro используется блок членства, который накладывает ограничения на все вложенные блоки в зависимости от уровня членства. Со временем плагины со сложными пользовательским интерфейсом клиентской части будут комплектоваться блоками, обеспечивающими большую степень контроля над макетом и настройками пользовательского интерфейса. Так, вы могли бы выполнять тонкую настройку макета каталога в BuddyPress с помощью блоков и их параметров.

Создание собственных блоков

Как бы ни было приятно использовать готовые блоки, создавать собственные блоки еще приятнее... или, скажем, почти всегда приятнее. Иногда это довольно трудно, однако давайте начнем с простейшего примера блока, после чего рассмотрим что-то посложнее.

Пример простейшего блока

Чтобы добавить блок в редакторе, нужно, как минимум, вызвать функцию `wp.blocks.registerBlockType()` в JavaScript-коде, запускаемом на странице редактора, что иллюстрирует следующий пример.

Пример

```
wp.blocks.registerBlockType('bawwp/minimal', {
  title: 'Minimal Block Example',
  category: 'common',
  edit() {
    return 'Minimal block editor content.';
  },
  save() {
    return 'Minimal block frontend content.';
  },
});
```

Полное описание и список параметров этой функции можно найти в разделе "Регистрация блоков" Руководства по редактору блоков ([oreil.ly/Ydx9m](https://core.trac.wordpress.org/browser/trunk/src/wp-admin/js/block-editor.js)).

Наш простейший блок передает в функцию имя блока и массив аргументов. Блокам следует давать уникальные имена, указывая в качестве префикса пространство имен или слаг того плагина, который загружает блок. Имена должны начинаться с буквы и могут содержать *строчные* буквы и цифры, а также дефисы.

Наш простейший блок также передает в функцию заголовок (`title`) и имя категории (`category`). В качестве заголовка допускается любая описательная строка, которая упростит поиск блока в общем списке. Чуть позже будет показано, как можно добавить новую категорию для группировки родственных блоков. В ядро WordPress включены следующие категории: *common* (общие), *formatting* (форматирование), *layout* (макет), *widgets* (виджеты) и *embed* (встроенные).

"Рабочими лошадками" функции `registerBlockType` являются атрибуты `edit` и `save`, в качестве которых передаются функции для вычисления и возвращения структуры блока. Функция `edit` используется при отображении блока в редакторе (рис. 11.2). Функция `save` вызывается при сохранении поста, перед сериализацией блоков в переменную `post_content`.

В данном простейшем примере блока мы возвращаем лишь простую строку. Обратите внимание, что при наличии в этой строке HTML-кода он будет санитирован и преобразован в HTML-сущности. Для генерирования разметки, содержащей ваши блоки, вам потребуется воспользоваться функцией `wp.element.createElement()`.

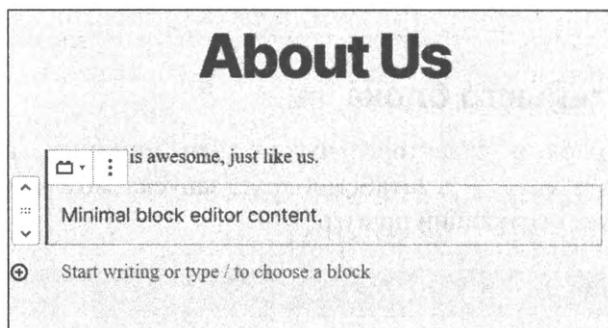


Рис. 11.2. Простейший блок в редакторе

Мы можем создать простой плагин с одним каталогом и двумя файлами, выполняющими загрузку нашего блока: `block.js` и `minimal-block-example.php`.

В следующем примере показано, как выглядит код файла `minimal-block-example.php`, выполняющий подключение файла `block.js`.

Пример

```
/**
 * Имя плагина: Простейший пример блока
 */
function enqueue_min_block() {
    wp_enqueue_script(
        'minimal-block',
        plugins_url('block.js', __FILE__),
        array('wp-blocks')
    );
}
add_action('enqueue_block_editor_assets', 'enqueue_min_block');
```

Данный код аналогичен коду для подключения другого JavaScript-кода, приводившемуся в *главе 9*. В рассмотренном случае вместо обычного хука действия `wp_enqueue_scripts` мы подключаемся к хуку действия `enqueue_block_editor_assets`. Также обратите внимание, что в качестве зависимости для JavaScript-кода нашего блока указывается пакет `wp-blocks`. По мере расширения функциональности блоков нам потребуется подключать здесь и другие пакеты, имеющие отношение к редактору Gutenberg.



Приведенный пример также демонстрирует минимальные требования к плагину. Единственный обязательный параметр в заголовке — имя плагина. Остальные параметры являются опциональными, хотя, как было показано в *главе 3*, присутствие некоторых из них будет крайне желательным.

Теперь мы рекомендуем вам снова обратиться к документации, представленной на сайте WordPress.org, и ознакомиться с разделом "Создание вашего первого типа блоков" (oreil.ly/7qLtP). В этом разделе освещается добавление стилей, использование редактируемых полей, добавление панелей инструментов и настроек, а также создание динамически обновляемых блоков.

Использование кастомных блоков для разработки интерфейсов приложений

Допустим, что преподавателям, работающим с нашим приложением SchoolPress, иногда требуется создавать новые домашние задания. Конечно, они могут использовать Microsoft Word или Adobe Acrobat или просто ввести нужный текст в виде обычного поста WordPress. Однако нам нужно нечто хорошо согласующееся с остальной частью создаваемого нами приложения, позволяющее сохранять ответы на задания в базе данных для обеспечения отчетности и другой функциональности.

Мы можем создать форму, написав кастомный PHP-код. Можно создать что-то более динамичное с помощью фреймворка React, обеспечив сохранение данных в WordPress посредством WordPress REST API. Также можно использовать плагин Advanced Custom Fields в сочетании с кастомным типом поста для домашнего задания, обеспечивающим надлежащую компоновку необходимых данных. Как и в случае многих других задач разработки WordPress-приложений и программной разработки в целом, в данном случае имеется несколько рационально обоснованных вариантов решения.

Наибольший интерес при этом вызывает вариант, подразумевающий создание кастомного типа поста для домашнего задания, кастомных типов и шаблонов блоков. Взяв за основу редактор блоков, мы будем базироваться на интерфейсе, уже знакомом для наших пользователей. Мы можем определить свои кастомные типы и шаблоны блоков таким образом, чтобы данные домашнего задания всегда представлялись в нужном для нашего WordPress-приложения структурированном формате.

Активация редактора блоков для кастомных типов постов

В версии 5.3 системы WordPress для кастомных типов постов по умолчанию задан классический редактор. Со временем для всех кастомных типов постов по умолчанию будет назначен новый редактор блоков. На данный момент для использования редактора блоков для кастомных типов постов необходимо его явно активировать при регистрации кастомных типов постов.

Давайте снова вернемся к нашему примеру с кастомным типом поста для домашнего задания. В следующем примере показана настройка редактора блоков.

Пример

```
register_post_type(
    'homework',
    array(
        'labels' => array(
            'name' => __('Homework'),
            'singular_name' => __('Homework')
        ),
        'public' => true,
        'has_archive' => true,
        'supports' => array('title', 'editor'), // Новая строка
        'show_in_rest' => true,                // Новая строка
    )
);
```

Строка `'supports' => array('title', 'editor')` добавляет поддержку для поля заголовка и редактора блоков. Поскольку редактор блоков использует REST API для обновления постов, мы также добавляем поддержку для REST API с помощью строки `'show_in_rest' => true`.

Категории блоков

При добавлении нескольких родственных блоков часто полезно отнести их к одной категории, чтобы они отображались рядом в списке блоков. Чтобы задать категорию блока, достаточно просто изменить атрибут `category`, передаваемый функции `registerBlockType()` внутри JavaScript-кода этого блока. В то же время о появлении новой категории блоков нужно проинформировать и систему WordPress. Для этого нужно воспользоваться фильтром `block_categories`, как показано в следующем примере.

Пример

```
// Добавляем категорию блоков для домашних заданий
function my_block_categories($categories, $post) {
    return array_merge(
        $categories,
        array(
            array(
                'slug' => 'homework',
                'title' => 'Homework',
            ),
        )
    );
}

add_filter('block_categories', 'my_block_categories', 10, 2);
```

Каждая категория в массиве `$categories` представляет собой массив с двумя ключами: `slug` и `title` (рис. 11.3).

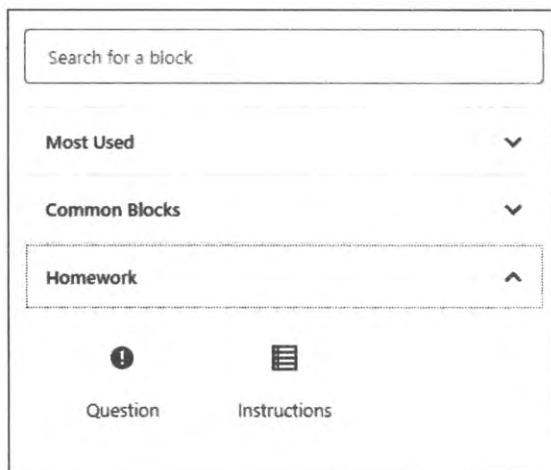


Рис. 11.3. Категория блоков Homework с блоками Homework и Instructions

Блоки домашнего задания

Для поддержки своего приложения для домашних заданий мы добавим пару блоков. Блок инструкций **homework/instructions** будет располагаться в верхней части нашего домашнего задания; помимо прочего, он будет содержать срок сдачи, задаваемый с помощью настроек блока. Ниже будут располагаться несколько блоков вопроса **homework/question**. Каждый блок вопроса будет иметь поле в формате Rich Text для ввода вопроса, тип которого будет задаваться с помощью настроек блока. Вопросы могут быть трех типов: с ответом "истина" или "ложь", с несколькими вариантами ответа, с развернутым ответом.

Полный код для этих блоков можно найти в папке для главы 11 в GitHub-репозитории этой книги (oreil.ly/mrmH0).

Ограничение типа блоков до определенных кастомных типов постов

Нам нужно, чтобы наши блоки для инструкций и вопросов использовались только в постах домашнего задания. Мы можем добиться этого, добавив код, который будет проверять тип редактируемого в данный момент поста и отменять регистрацию наших типов блоков, если это не пост домашнего задания.

В нижней части файла `block.js` добавьте фрагмент, приведенный в следующем примере.

Пример

```
// Отменяем регистрацию инструкций, если это не пост домашнего задания
wp.domReady(function() {
    if(wp.data.select('core/editor').getCurrentPostType() != 'homework') {
        wp.blocks.unregisterBlockType('homework/instructions');
    }
});
```

Подобно функции `document.ready()` библиотеки jQuery, функция `wp.domReady()` ожидает, пока завершится полная загрузка страницы, а затем выполняет определенный внутри нее код. Нам необходимо это сделать по той причине, что информация о типе редактируемого в данный момент поста становится доступной только после загрузки страницы. Именно поэтому мы вынуждены отменять регистрацию блоков, а не проверять тип поста еще до их регистрации. Для использования функции `wp.domReady()` убедитесь в том, что JavaScript-код вашего блока подключается с указанием в качестве зависимости пакета `wp-dom-ready`.

Тип текущего поста извлекается из модуля данных WordPress с помощью команды `wp.data.select('core/editor').getCurrentPostType()`. При этом следует убедиться в том, что JavaScript-код вашего блока подключается с указанием в качестве зависимости пакета `wp-edit-post`.

В "Справочнике по модулю данных" (oreil.ly/VPIbS) можно найти массу полезной информации по применению аналогичных методов. Для получения более подробных сведений о том, как устроен модуль данных, и как можно определять собственные хранилища данных, ознакомьтесь с документацией по модулю данных (oreil.ly/Q22Sb).

Ограничение кастомного типа постов до определенных блоков

Ранее было показано, как можно исключить использование наших блоков в постах других типов. В WordPress также имеется фильтр `allowed_block_types`, позволяющий разрешить применение только избранных типов блоков в рамках всего вашего сайта или сети. Также можно проверять, чему равно свойство `post_type` переданного объекта `$post`, и разрешать использование только избранных типов блоков при редактировании постов определенного типа.

Код, приведенный в следующем примере, ограничивает использование постов домашнего задания до нескольких избранных блоков ядра и наших блоков для инструкций и вопросов.

Пример

```
// Разрешаем использовать только определенные блоки в постах домашних заданий
function my_allowed_block_types($allowed_blocks, $post) {
    if ($post->post_type == 'homework') {
```

```

    $allowed_blocks = array(
        'core/block',
        'core/image',
        'core/paragraph',
        'core/heading',
        'core/list',
        'homework/instructions',
        'homework/question',
    );
}
return $allowed_blocks;
}
add_filter('allowed_block_types', 'my_allowed_block_types', 10, 2);

```

По умолчанию массив `$allowed_blocks` пуст, из чего WordPress может заключить, что разрешены все типы блоков. Поскольку блоки добавляются в JavaScript-коде после загрузки всего PHP-кода, фильтр не может знать, какие кастомные типы блоков будут добавлены. Однако, задав список допустимых типов блоков, как показано в данном примере, вы можете гарантировать, что будут загружены только блоки, относящиеся к указанным в списке типам.

Шаблоны блоков

При регистрации кастомного типа постов можно передать атрибут `template` для указания группы блоков, добавляемой в новые посты этого типа по умолчанию. В следующем примере представлена возможная версия нашего обратного вызова, которая регистрирует тип поста для домашнего задания, по умолчанию содержащий блок инструкций в верхней части, а ниже его — один вопрос с ответом "истина" или "ложь" и один вопрос с развернутым ответом.

Пример

```

// Регистрируем кастомный тип постов для домашних заданий
register_homework_post_type(
    'homework',
    array(
        'labels' => array(
            'name' => __('Homework'),
            'singular_name' => __('Homework')
        ),
        'public' => true,
        'has_archive' => true,
        'supports' => array('title', 'editor'),
        'show_in_rest' => true,
        'template' => array(
            array('homework/instructions'),

```

```

        array('homework/question',
              array('content' => 'True/false 1.',
                    'question_type' => 'true_false'
              )
        ),
        array('homework/question',
              array('content' => 'Essay question.',
                    'question_type' => 'essay'
              )
        ),
    ),
);

```

После добавления кода из этого примера новые посты домашних заданий будут создаваться на основе нашего шаблона, сразу получая блок инструкций и пару блоков вопросов. Чтобы перейти к редактированию того или иного блока, нужно будет щелкнуть по нему мышью (рис. 11.4).

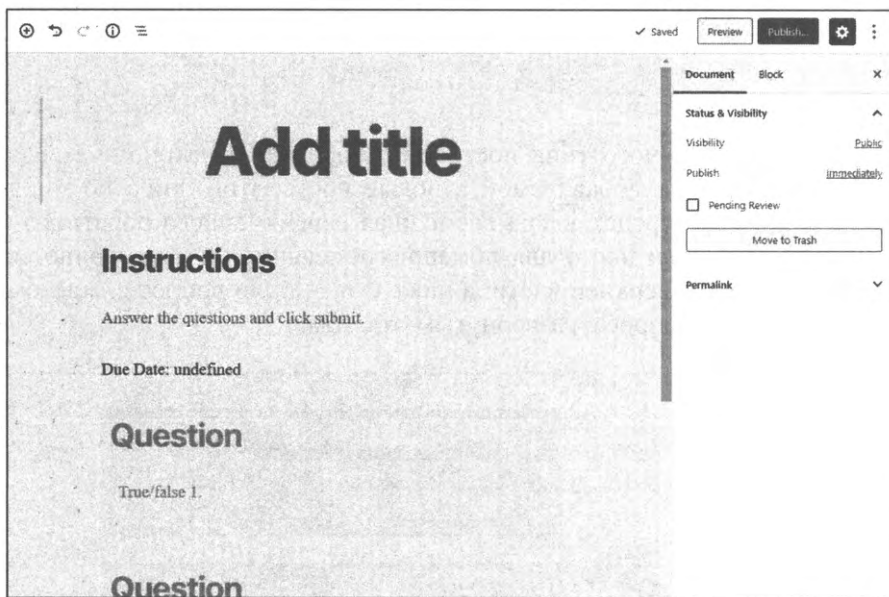


Рис. 11.4. Заданный по умолчанию шаблон новых постов для домашних заданий

Вы также можете добавлять шаблоны для базовых типов постов или других кастомных типов постов после их регистрации, добавлять шаблоны в JavaScript-коде своего блока, блокировать шаблоны во избежание их перенастройки или удаления, а также вкладывать шаблоны друг в друга. Описание этих и других возможностей можно найти в разделе "Шаблоны блоков" Руководства по редактору блоков (oreil.ly/OEjyi).

Сохранение данных блока в метаданных поста

По умолчанию атрибуты блоков сохраняются в специально отформатированных HTML-комментариях в теле `post_content`. Например, наш блок инструкций может выглядеть примерно так, как показано в следующем примере, перед его преобразованием для клиентской части.

Пример

```
<!-- wp:homework/instructions -->
<p class="wp-block-homework-instructions-content">Email me your answers.</p>
<p class="wp-block-homework-instructions-due_date">Due Date: 2020-11-01</p>
<!-- /wp:homework/instructions -->
```

Если наш блок будет настроен правильно, то его содержимое и срок сдачи можно будет изменять с помощью редактора блоков и отображаемой для данного блока панели настроек. После внесения этих изменений они будут отображаться в клиентской части сайта.

Допустим, что нам требуется сортировать домашние задания по сроку их сдачи или создавать другие отчеты на основе данных, задаваемых в наших блоках. Эта задача будет вполне осуществимой, если интересующие нас данные будут сохраняться в метаданных поста.

Чтобы привязать атрибуты блока к метаданным поста, мы должны зарегистрировать поля метаданных нашего поста. Поскольку мы не всегда регистрируем метаданные постов, вы можете просто вызвать функцию `update_post_meta()` напрямую и добавить нужные вам данные. Однако если манипуляции над метаданными поста осуществляются с помощью REST API, потребуется зарегистрировать и этот интерфейс, как показано в следующем примере.

Пример

```
register_post_meta('homework', '_homework_due_date', array(
    'show_in_rest' => true,
    'single' => true,
    'type' => 'string',
    'auth_callback' => function() {
        return current_user_can('edit_posts');
    }
));
```

После этого можно будет выполнять привязку к метаданным поста, просто указывая источник в атрибуте `source` блока, что иллюстрирует следующий пример.


```

wp.blocks.registerBlockType('homework/instructions', {
    // ...

    attributes: {
        content: {
            type: 'array',
            source: 'children',
            selector: 'p',
        },
        due_date: {
            type: 'string',
            meta: '_homework_due_date',
            source: 'meta',
            default: '',
        }
    },
    // ...
}

```

Более подробные сведения о том, как можно редактировать метаданные постов с помощью блоков, можно найти в разделе "Сохранение метаданных поста с помощью блока" Руководства по редактору блоков (oreil.ly/xGpPW).

Советы

Работа с редактором Gutenberg — сложная и подверженная частым изменениям тема, для освещения которой требуется отдельная книга. Мы рассмотрели здесь базовые возможности и несколько приемов, которые могут оказаться полезными для использующих этот редактор разработчиков приложений. Далее также приводится ряд общих советов, которые могут оказаться вам полезными, и указывается несколько источников дополнительной информации.

Активируйте режим отладки с помощью константы `WP_SCRIPT_DEBUG`

По умолчанию WordPress минимизирует версии JavaScript-скриптов, размещая весь JavaScript-код в одном файле. Это в значительной мере затрудняет вывод в консоли сообщений об ошибках JavaScript-кода. При выполнении любой JavaScript-разработки, и в особенности при работе с редактором Gutenberg, следует активировать режим отладки с помощью константы `WP_SCRIPT_DEBUG`.

Добавив строку `define('WP_SCRIPT_DEBUG', true);` в свой файл `wp-config.php`, вы дадите системе WordPress указание использовать полные версии скриптов и загружать все JavaScript-файлы по отдельности. При этом отображаемые в консоли сообщения об ошибках будут давать удобные ссылки на имена файлов и номера строк, которые пригодны для поиска соответствующего исходного кода.

Задавайте версию скрипта с помощью функции *filemtime()*

При подключении JavaScript-скриптов вы можете указывать их версию. Поскольку браузеры активно кэшируют JavaScript-файлы, вы можете обновлять версию скриптов, чтобы прерывать это кэширование. При наличии JavaScript-файлов вам придется часто их редактировать и обновлять. Один из способов не допустить кэширования браузером JavaScript-файла в тот момент, когда вы его редактируете, состоит в том, чтобы указывать в качестве версии дату последнего изменения.

В следующем примере показано, как можно задавать версию скрипта с помощью функции `filemtime()` в коде для подключения наших блоков вопросов.

Пример

```
wp_register_script(
    'homework-question',
    BWAWWP_URL . 'homework-cpt/blocks/question/blocks.js',
    array('wp-blocks',
          'wp-element',
          'wp-editor',
          'wp-components',
          'wp-dom-ready',
          'wp-edit-post',
    ),
    filemtime(BWAWWP_DIR . 'homework-cpt/blocks/question/blocks.js')
);
```

Перед разворачиванием этого кода в виде плагина вам потребуется обновить версию скрипта, указав конкретный номер версии.

Дополнительные советы

Ряд отличных советов, вместе с описанием "продвинутых" приемов работы с блоками, предлагает Зак Гордон в своей статье "31 совет по работе с редактором Gutenberg" (oreil.ly/cS0Lk).

Глубже изучите JavaScript, Node.js и React

Более глубокое понимание "ванильного" языка JavaScript, Node.js и React станет хорошим подспорьем в работе с блоками в WordPress. Уделите какое-то время тому, чтобы что-то почитать о каждой из этих технологий и попробовать применить их на практике, чтобы хотя бы в общих чертах понимать, что они собой представляют. Вы обнаружите, что отладка проблем при работе с редактором Gutenberg представляет гораздо меньше трудностей, когда вы понимаете, какой слой вашего стека является источником проблемы.

Далее приведены рекомендации по дальнейшему изучению этих технологий.

◆ JavaScript

О языке JavaScript и истории его применения в WordPress было рассказано в *главе 9*. Чем основательнее будет ваше понимание языка JavaScript, тем легче вам будет даваться разработка блоков. Прекрасными пособиями по этому языку являются книга Дэвида Флэнагана *JavaScript: The Definitive Guide* и книга Дугласа Крокфорда *JavaScript: The Good Parts*. Вы можете кратко ознакомиться с ними, чтобы почерпнуть основы, а затем использовать их в качестве справочника или для более глубокого "погружения" в JavaScript.

◆ Node.js

Node.js — это среда выполнения JavaScript-кода, которая работает в вашем серверном окружении. Конфигурация компоновки при работе с редактором Gutenberg обычно включает в себя ряд инструментов на базе Node.js, в том числе Webpack, Babel, и Node Package Manager (npm). Упаковщик модулей Webpack позволяет вам объединить весь свой JavaScript-код в единый пакет для его запуска в браузере. Babel позволяет преобразовать код, написанный на диалектах ESNext и JSX, в JavaScript-код, работающий в большинстве современных браузеров. Менеджер пакетов npm служит для установки JavaScript-пакетов и управления зависимостями между ними. И Webpack, и Babel обычно устанавливаются как модули Node.js. Иногда вам также потребуется использовать в JavaScript-коде своих блоков дополнительные библиотеки Node.js. Провести их установку и осуществлять управление ими можно будет с помощью npm. В разделе "Конфигурация компоновки JavaScript-кода" Руководства по редактору блоков (oreil.ly/4AbjU) вы найдете пошаговые инструкции по установке Node.js и npm, а также последующей установке пакета *@wordpress/scripts*, который включает в себя рекомендуемые дефолтные конфигурации для Webpack и Babel.

◆ React

Фреймворк React был подробно рассмотрен в *главе 9*. В двух словах можно сказать, что React — это JavaScript-библиотека для создания интерактивных пользовательских интерфейсов, которые часто присутствуют на экранах редактора блоков. Блок системы WordPress можно рассматривать как набор компонентов пользовательского интерфейса фреймворка React, созданный для реализации стандартного редактора или определенного стиля контента. WordPress располагает собственной библиотекой React-компонентов, предназначенных для создания блоков. Вы можете комбинировать их с другими React-компонентами или со своими кастомными компонентами. Попробуйте использовать React не только для создания блоков редактора Gutenberg — это даст вам более четкое представление о React в целом. Столкнувшись с проблемами при создании блоков в WordPress, вы быстрее сможете понять, что является источником проблемы — React или WordPress, и благодаря этому быстрее получить необходимую помощь. Если вы хотите получше ознакомиться с React, прочитайте книгу *Learning React* под авторством Алекса Бэнкса и Евы Порселло. Также будет очень полезно ознакомиться с официальным руководством, представленным на сайте фреймворка React (oreil.ly/ITO6k), или одним из имеющихся там примеров (reactjs.org/community/examples.html).

Многосайтовые сети в WordPress

С выходом версии 3.0 в WordPress появился многосайтовый режим, или "многосайтовость" (WordPress Multisite). Ранее эта функциональность разрабатывалась как многопользовательская версия WordPress в рамках отдельного проекта с открытым исходным кодом (WordPress Multiuser, WPMU). Поскольку значительная часть кода WPMU и WordPress совпадала, было принято решение объединить эти два проекта. Многосайтовость дает администраторам WordPress возможность создать собственную многосайтовую сеть. Все сайты такой многосайтовой сети совместно используют одну и ту же базу данных и одни и те же файлы исходного кода. При каждом создании в сети нового сайта в базе данных для него формируются новые таблицы.

Когда целесообразна многосайтовость?

Если вы используете несколько экземпляров WordPress, особенно если они служат основой для достаточно похожих сайтов, то, возможно, вам стоит воспользоваться многосайтовым режимом, чтобы сэкономить свое время и деньги. Подумайте о том, насколько проще обновлять все свои экземпляры WordPress одновременно, вместо того чтобы обновлять их по отдельности. Вот лишь некоторые преимущества использования многосайтового режима (рис. 12.1):

- ◆ Общий набор плагинов, тем и кастомного кода на нескольких сайтах.
- ◆ Управление всеми пользователями вашей сети в одном месте.
- ◆ Доступ ко всем вашим сайтам с помощью одного аккаунта суперадминистратора.
- ◆ Одновременное обновление ядра WordPress и установленных плагинов и тем в одном месте вместо обновления их на нескольких сайтах.
- ◆ Простое развертывание нового сайта с помощью пары щелчков мыши вместо настройки "с нуля" нового экземпляра WordPress.
- ◆ При наличии фреймворка дочерних тем на всех сайтах сети, можно одновременно обновлять все свои темы, используя хуки фреймворка тем.
- ◆ Предоставление пользователям возможности создавать и администрировать собственные сайты в вашей сети. Администратор одного сайта может иметь другую роль на другом сайте.

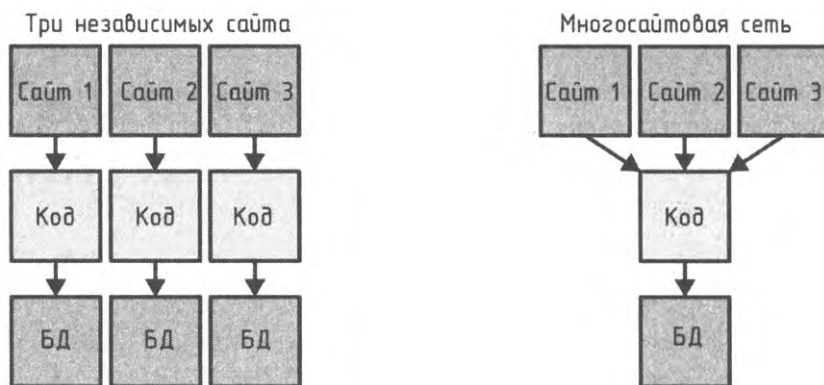


Рис. 12.1. Многосайтовый режим в сравнении с наличием самостоятельных сайтов

Вот несколько примеров ситуаций, хорошо подходящих для реализации многосайтового режима WordPress:

- ◆ Вам нужно создать сеть аналогичных сайтов таким образом, чтобы администратор каждого сайта управлял контентом своего сайта. Допустим, вам требуется создать сеть школьных сайтов для школьного округа так, чтобы все школы имели собственные сайты с практически одинаковым набором функций. Администратор каждого сайта должен располагать возможностью обновления всего контента и символики на сайте. В таком случае многосайтовый режим WordPress — именно то, что вам нужно!
- ◆ Вам необходим многоязычный сайт с несколькими администраторами, и вы не питаете особого доверия к имеющемуся плагину для создания многоязычных WordPress-сайтов (WPML). Воспользуйтесь многосайтовым режимом WordPress!
- ◆ Вам нужно создать сотни дочерних сайтов с легким процессом администрирования и добавления новых элементов. И здесь снова прекрасно подойдет многосайтовый режим WordPress!

Когда лучше отказаться от многосайтовости?

Многие из аргументов в пользу многосайтового режима одновременно могут служить и аргументом против его использования. Так, иногда одновременное управление всеми пользователями сети может привести к путанице или проблемам правового характера. В некоторых случаях наличие нескольких версий определенных тем, плагинов или самой системы WordPress является неотъемлемой необходимостью.

Еще одним аргументом против многосайтовости могут стать проблемы с использованием сторонних плагинов. Хотя большинство плагинов хорошо сочетается с многосайтовым режимом WordPress, некоторые плагины могут работать не так, как

ожидалось. Так, например, плагин, модифицирующий вид экрана для добавления нового пользователя в панели администратора, может не принять во внимание тот факт, что в режиме многосайтовости этот экран будет выглядеть иначе. Или может получиться так, что плагин будет сохранять данные в рамках всей сети вместо того, чтобы делать это в отдельности для каждого блога. Опять же, повторим, что большинство плагинов не доставит вам никаких проблем, но время от времени вам будет попадаться плагин, который нужно будет исправить, заменить или модифицировать для сочетания с вашей конкретной многосайтовой конфигурацией.

В общем, мы не рекомендуем вам проводить апгрейд своей установки WordPress до многосайтовой версии, если вам не нужно наличие нескольких сайтов. А если вам требуется использовать несколько сайтов, рассмотрите все преимущества и недостатки их размещения в единой многосайтовой сети.

Альтернативы многосайтового режима

Далее перечислен ряд альтернативных решений, которые вам стоит рассмотреть перед тем, как применять многосайтовый режим для своего WordPress-проекта.

Множество авторов или категорий на одном и том же WordPress-сайте

Если вам нужно просто организовать свой контент в виде отдельных "сайтов" или "разделов", подумайте об использовании нескольких авторов блогов или нескольких категорий на одном и том же WordPress-сайте. При наличии нескольких авторов блогов на одном сайте вы можете модифицировать свою тему так, чтобы архив постов каждого автора обладал собственным стилем и выглядел как отдельный сайт.

Если вам необходимо выделить некоторый второстепенный контент в отдельный блог, можно создать отдельную категорию постов для этого контента. Применяя фильтры, можно не допустить включения этого контента в основной индекс. Также можно создать ссылку на представление архива постов этой категории и снабдить его собственным стилем, чтобы он выглядел как "блог" в рамках более крупного сайта.

Кастомные типы постов

Организовать контент так, чтобы создавалось впечатление отдельных сайтов без реального создания таковых, также можно с помощью кастомных типов постов. Как и в случае с использованием нескольких категорий постов, вы можете фильтровать основной индекс, чтобы исключить попадание в него определенных кастомных типов постов и обрабатывать представление архива определенных кастомных типов постов как отдельный сайт.

Абсолютно самостоятельные сайты

Если один и тот же контент вам нужно поместить на нескольких сайтах или обеспечить их бесшовное взаимодействие, то стоит подумать об их размещении в единой многосайтовой сети. Еще один подход состоит в том, чтобы создать отдельные сайты и задействовать REST API или такие инструменты, как разработанный нами плагин WordPress Single Sign-On (SSO), для объединения контента и пользователей в рамках всех сайтов. (REST API и плагин SSO были подробно рассмотрены в *главе 10*.)

Сервис обслуживания WordPress-сайтов

Если вас главным образом привлекает возможность обновления тем и плагинов WordPress из одного места, подумайте об использовании сервиса обслуживания Wordpress-сайтов. Существует много популярных решений такого рода. Так, в частности, возможность администрирования и обновления нескольких WordPress-сайтов из одной панели администратора предоставляют следующие приложения, плагины и сервисы:

- ◆ приложение Calypso компании Automattic (oreil.ly/l3-rM);
- ◆ сервис ManageWP (managewp.com);
- ◆ плагин MainWP (mainwp.com);
- ◆ плагин InfiniteWP (infinitewp.com).

Одновременно обновлять несколько WordPress-сайтов также можно с помощью интерфейса командной строки WP-CLI (wp-cli.org). Отличную статью об этом написал Фил Бэнкс (oreil.ly/18CCS).

Мультиарендность

Если вам нравится идея общего хранилища для системы WordPress, ее тем и плагинов, но вы не хотите создавать предлагаемую многосайтовым режимом WordPress архитектуру общей базы данных, то обратите свой взор на мультиарендные решения. В упрощенном виде мультиарендность в WordPress выглядит так: вы создаете символическую ссылку на каталог ядра WordPress и используете эти файлы сразу в нескольких экземплярах WordPress на одном сервере или кластере серверов. На практике для того, чтобы WordPress "подружился" с такой конфигурацией, потребуется приложить определенные усилия. Хорошей отправной точкой для более глубокого изучения темы мультиарендности в WordPress является лекция Клиффа Сила на канале WordPress.tv (oreil.ly/Svf-V).

Настройка многосайтовой сети

После того как вы установите, что многосайтовая сеть — именно то, что вам нужно, вам потребуется ее настроить. Процесс настройки многосайтовости достаточно прост, хотя и выглядит чуть сложнее, чем задание определенного параметра в па-

нели администратора. Если речь не идет о настройке нового экземпляра WordPress, то прежде всего вам потребуется провести резервное копирование базы данных и каталога с файлами.

Откройте файл `wp-config.php` в корневом каталоге вашего WordPress-приложения и добавьте следующую строку кода после строки `/* That's all, stop editing! Happy blogging. */`:

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

Обновите панель администратора WordPress и наведите указатель мыши на пункт меню **Tools** (Инструменты). В открывшемся подменю щелкните по кнопке **Network Setup** (Настройка сети). Откроется страница **Network Setup** (Настройка сети) с полями для ввода следующей информации:

- ♦ **Поддомен или подкаталог.** Решите, как должны быть организованы подсайты в вашей многосайтовой сети. Если они должны представлять собой поддомены вида `sub.domain.com`, выберите вариант **subdomain (поддомен)**. Если они должны представлять собой подкаталоги вида `domain.com/sub`, выберите вариант **subdirectory (подкаталог)**. Вы всегда можете воспользоваться плагином для отображения доменов и получить любое нужное вам доменное имя, будь то поддомен или подкаталог.
- ♦ **Название сети** — название вашей многосайтовой сети.
- ♦ **Адрес электронной почты администратора** — адрес электронной почты администратора сети, т. е. ваш адрес электронной почты.

После ввода запрашиваемой информации нажмите кнопку **Install** (Установить). Вы увидите два текстовых поля. Первое текстовое поле содержит код, который вам нужно скопировать и вставить в файл `wp-config.php`, между строкой `/* That's all, stop editing! Happy blogging. */` и той строкой кода, которую вы добавили ранее:

```
define('MULTISITE', true);
define('SUBDOMAIN_INSTALL', false);
define('DOMAIN_CURRENT_SITE', 'whatever.com');
define('PATH_CURRENT_SITE', '/');
define('SITE_ID_CURRENT_SITE', 1);
define('BLOG_ID_CURRENT_SITE', 1);
```



Под многосайтовостью в WordPress понимается единая *сеть* с множеством *подсайтов* в рамках этой сети. На данный момент понятие *блог* уже не рекомендуется, однако оно по-прежнему присутствует в относящихся к многосайтовости таблицах базы данных и функциях. Так, для идентификации сети служит идентификатор `site_id`, а для идентификации подсайта — идентификатор `blog_id`. Хотя слово "*подсайт*" для обозначения дочернего сайта в рамках сети в какой-то мере помогает избежать путаницы, вам все же придется научиться определять, какая разновидность "сайта" имеется в виду в том или ином фрагменте кода или документации.

В данном примере мы решили создать подкаталоги и потому присвоили константе `SUBDOMAIN_INSTALL` значение `false`. Если бы мы захотели использовать поддомены, то присвоили бы константе `SUBDOMAIN_INSTALL` значение `true`.

Код, содержащийся во втором текстовом поле, необходимо скопировать и вставить в файл *.htaccess, который также расположен в корневом каталоге вашего экземпляра WordPress.

Пример

```
RewriteEngine On
RewriteBase /
RewriteRule ^index\.php$ - [L]

# добавляем косую черту после /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).*) $2 [L]
RewriteRule ^([_0-9a-zA-Z-]+)/?(.*\.php)$ $2 [L]
RewriteRule . index.php [L]
```



Поскольку вы можете использовать кастомный файл *.htaccess (в силу того, что хосту или вам самим нужно делать некоторые специфические вещи), здесь необходимо убедиться, что вы замещаете дефолтные правила перезаписи WordPress. Также можно заметить, что правила перезаписи, создаваемые для поддоменов, отличаются от правил, создаваемых для подкаталогов.

После того как вы скопируете код обоих текстовых полей в соответствующие файлы и сохраните эти файлы на своем веб-сервере, нужно будет обновить окно браузера.

Вам будет предложено пройти авторизацию еще раз. Авторизуйтесь, указав логин и пароль своей учетной записи администратора. Вот и все! Теперь вы управляете многосайтовой сетью WordPress в качестве суперадминистратора и имеете полный доступ ко всем сайтам вашей сети (в то время как обычные администраторы имеют полный доступ только к своему сайту). Будьте крайне осмотрительны в отношении того, кому вы предоставляете права суперадминистратора.

Если вы решили использовать поддомены, а не подкаталоги, то вам следует выполнить еще пару шагов, которые сэкономят вам время в долгосрочной перспективе. Определите запись шаблонной маски поддомена на странице настройки DNS-сервера регистратора доменов. Если вы помните, при первой установке WordPress вы определили запись с типом А (хост) и IP-адресом вашего аккаунта хостинга. Теперь в том же месте в качестве имени хоста введите символ *, а не символ @ и укажите тот же IP-адрес, который был указан в записи с символом @. Эта запись будет охватывать все поддомены вашего основного домена. Таким образом, и поддомен **whatever.whatever.com**, и поддомен **somethingelse.whatever.com** будут одинаково отображаться на тот же IP-адрес, что и основной домен.

В зависимости от того, какой у вас аккаунт хостинга, иногда также требуется определить и запись шаблонной маски поддомена, указывающую на тот же каталог, на

который указывает ваш основной домен. Для этого зарегистрируйте новый поддомен вида ***.<whatever>.com** и сопоставьте его с тем же каталогом, с которым сопоставлен домен **<whatever>.com**.

Зачем же мы все это делаем? Мы задаем шаблонную маску поддоменов для того, чтобы при создании новых сайтов в многосайтовой сети их можно было использовать сразу после их создания. Если мы этого не сделаем, то нам потребуется вручную добавлять каждый создаваемый нами поддомен на странице настройки регистратора доменов и хоста. Но если мы всего один раз настроим шаблонную маску поддоменов, она будет автоматически срабатывать для каждого поддоменного сайта, создаваемого нами в многосайтовой сети.



В зависимости от того, какой хост вы используете, вам все же иногда потребуется выполнить ряд действий вручную на стороне сервера. Если вы работаете с панелью **cPanel** и хотите сопоставить некоторый домен со своим многосайтовым экземпляром WordPress, добавьте "дополнительный домен" (Addon Domain), сопоставив его с корневым каталогом вашей многосайтовой системы.

Администрирование многосайтовой сети

Чтобы перейти к администрированию созданной вами многосайтовой сети, выберите в меню администратора WordPress пункт **My Sites → Network** (Мои сайты → Администратор сети). Попасть в панель администратора сети также можно, перейдя по адресу **<whatever>.com/wp-admin/network/**. Панель администратора сети выглядит во многом так же, как панель администратора любого сайта в рамках этой сети. Чтобы не перепутать ее с другой панелью, убедитесь, что адрес в адресной строке браузера заканчивается сегментом **/network/**.

Панель администратора

Панель администратора сети выглядит почти так же, как исходная панель администратора стандартного экземпляра WordPress, отличаясь лишь наличием ссылок для быстрого добавления новых сайтов и пользователей сети в виджете **Right Now** и двух дополнительных текстовых полей для поиска конкретных пользователей и сайтов. Как и в случае обычной панели администратора WordPress, вы можете полностью кастомизировать сетевую панель с помощью плагинов или кастомного кода.

Сайты

На странице управления сайтами вы можете управлять всеми сайтами своей сети. Вы можете добавить любое нужное вам количество сайтов, и даже предоставить пользователям сети возможность создавать собственные сайты.

Добавление нового сайта не представляет каких-либо трудностей. Для этого нужно щелкнуть по кнопке **Add New** (Добавить новый) в верхней части страницы или выбрать пункт меню **Sites → Add New** (Сайты → Добавить новый).

На открывшейся странице **Add New Site** (Добавление нового сайта) вы увидите следующие поля:

- ♦ **Site address** (Адрес сайта) — в зависимости от того, как вы настроили сеть, здесь нужно ввести либо поддомен, либо подкаталог сайта.
- ♦ **Site title** (Название сайта) — название вашего нового сайта.
- ♦ **Administrator email** (Адрес электронной почты администратора) — адрес электронной почты администратора вашего нового сайта. Здесь необязательно указывать свой адрес электронной почты; вы можете указать адрес клиента или пользователя, для которого вы настраиваете новый экземпляр WordPress.

Щелкните по кнопке **Add Site** (Добавить сайт), и вуаля! — вы получили новый WordPress-сайт. Как видите, данный процесс многократно короче по сравнению с настройкой "с нуля" нового экземпляра WordPress.

Пользователи

Все сайты, создаваемые вами в своей сети, будут иметь один и тот же пул пользователей. Фактически данные обо всех пользователях сохраняются в таблице `wp_users`, и каждый пользователь обладает метаданными, привязывающими его к одному или нескольким сайтам сети. На странице управления пользователями вы можете управлять всеми пользователями сети, назначать любых пользователей суперадминистратором с правами на управление всей многосайтовой сетью и получать информацию о том, членом каких сайтов является каждый пользователь.

Чтобы открыть страницу **Add New User** (Добавление нового пользователя), щелкните по кнопке **Add New** (Добавить нового) в верхней части страницы, либо выберите пункт меню **Users** → **Add New** (Пользователи → Добавить нового). На странице **Add New User** (Добавление нового пользователя) вы увидите следующие поля:

- ♦ **Username** (Имя пользователя) — логин создаваемого вами нового пользователя. Должен включать в себя только строчные буквы или цифры, без пробелов или специальных символов.
- ♦ **Email** (Адрес электронной почты) — адрес электронной почты создаваемого вами нового пользователя.

После того как вы щелкните по кнопке **Add User** (Добавить пользователя), добавленный вами пользователь получит электронное письмо с логином и паролем для авторизации на дефолтном сайте верхнего уровня с заданной вами для него ролью по умолчанию. Данный способ добавления пользователей нельзя назвать идеальным, поскольку при этом требуется выполнять дополнительный шаг добавления пользователя на конкретные подсайты сети. В зависимости от имеющейся ситуации, иногда проще добавлять новых пользователей на подсайт непосредственно из подсайта, где добавление новых пользователей осуществляется точно так же, как в случае обычного экземпляра WordPress. Если при попытке добавить пользователя на сайт окажется, что в сети уже присутствует введенный логин, будет выдано сообщение о том, что такой логин уже существует. Чтобы добавить на сайт такого

пользователя, введите его логин в разделе **Add Existing User** (Добавление существующего пользователя), выберите для него роль и укажите, следует ли отправлять ему электронное письмо для подтверждения.

Темы

Здесь перечислены все темы, имеющиеся в каталоге `/wp-content/themes/`. Вы можете управлять каждой из тем, доступных для сайтов вашей сети. Чтобы тема стала доступной для использования на отдельном сайте, ее необходимо активировать в рамках всей сети. Если вы этого не сделаете, то тема даже не появится в списке доступных для активации тем на странице **Appearance** → **Themes** (Внешний вид → Темы) отдельного сайта.

Плагины

Здесь перечислены все плагины, имеющиеся в каталоге `/wp-content/plugins/directory`. Вы можете активировать плагины в рамках всей сети, чтобы они автоматически запускались на всех сайтах сети, включая и новые сайты. Плагины, активированные в рамках всей сети, не отображаются на странице для управления плагинами отдельного сайта. На самом деле, если вы специально не активируете меню плагинов на странице настроек сети (см. следующий раздел), то администраторы отдельных сайтов даже не смогут открыть страницу плагинов.

Если вы предоставите администраторам отдельных сайтов возможность управления своими плагинами, то им будут доступны для активации только уже установленные плагины; они не смогут устанавливать собственные плагины. Это весьма разумно, поскольку вы должны знать, какие плагины доступны на каждом сайте сети. Вам не нужно, чтобы администратор отдельного сайта мог устанавливать какие угодно сторонние или кастомные плагины, поскольку это потенциально может плохо сказаться на других сайтах сети.

Для добавления нового плагина на уровне сети нужно добавить его точно таким же образом, как это делается в случае обычного экземпляра WordPress.



Не все плагины следует активировать на уровне сети. Принимая решение о том, как лучше активировать тот или иной плагин — на уровне сети или по отдельности на каждом сайте, — сверьтесь с документацией этого плагина. Общее правило сводится к тому, чтобы активировать плагин по отдельности на каждом сайте, если он требуется лишь на некотором подмножестве сайтов или требует независимого управления данными или кастомными типами постов в рамках каждого отдельного сайта.

Настройки

В отличие от обычных настроек экземпляра WordPress, задаваемых вами для стандартного WordPress-сайта, эти настройки действуют в рамках всей сети.

Раскрыв пункт **Settings** (Настройки) в расположенном слева меню администратора, вы увидите следующие подпункты:

◆ Рабочие настройки:

- название сети;
- адрес электронной почты администратора сети.

◆ Настройки регистрации:

- разрешение регистрации новых пользователей;
- уведомление о регистрации;
- добавление новых пользователей;
- запрещенные имена;
- ограничения регистрации с помощью адреса электронной почты;
- запрещенные почтовые домены.

◆ Настройки новых сайтов:

- приветственное электронное письмо;
- приветственное электронное письмо для пользователя;
- первый пост;
- первая страница;
- автор первого комментария;
- URL-адрес первого комментария.

◆ Настройки загрузки на сайт:

- размер пространства для загрузки на сайт;
- типы загружаемых на сайт файлов;
- максимальный размер загружаемого на сайт файла.

◆ Настройки меню:

- активировать меню администрирования.

Обновления

Как и в случае стандартного экземпляра WordPress, данная страница позволяет обновлять и ядро WordPress, и все требующие обновления плагины и темы. Прекрасной особенностью многосайтовой сети при этом является то, что вы можете выполнять сразу все обновления для всех сайтов сети. Это гораздо эффективнее по сравнению с выполнением обновлений для каждого отдельного экземпляра WordPress. Здесь вы одним махом обновляете WordPress, а также все плагины и темы!

Структура базы данных многосайтовой сети

Все сайты многосайтовой сети совместно обращаются к одной и той же базе данных. При активации многосайтового режима WordPress в вашей базе данных создается несколько новых таблиц.

Общесетевые таблицы

wp_site

В таблице wp_site содержится основная информация о многосайтовой сети, а именно: ее идентификатор, доменное имя и путь. Обычно эта таблица содержит только одну запись. Структура таблицы wp_site показана в табл. 12.1.

Таблица 12.1. Структура данных таблицы wp_site

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
id	bigint(20)		Нет	Нет	AUTO_INCREMENT
domain	varchar(200)	utf8_general_ci	Нет		
path	varchar(100)	utf8_general_ci	Нет		

wp_sitemeta

В таблице wp_sitemeta содержатся все общесетевые параметры и настройки. Структура таблицы wp_sitemeta показана в табл. 12.2.

Таблица 12.2. Структура данных таблицы wp_sitemeta

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
meta_id	bigint(20)		Нет	Нет	AUTO_INCREMENT
site_id	bigint(20)		Нет	0	
meta_key	varchar(255)	utf8_general_ci	Да	NULL	
meta_value	longtext	utf8_general_ci	Да	NULL	

wp_blogs

В таблице wp_blogs содержится информация о каждом отдельном сайте многосайтовой сети. Структура таблицы wp_blogs показана в табл. 12.3.

Таблица 12.3. Структура данных таблицы *wp_blogs*

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
blog_id	bigint(20)		Нет	Нет	AUTO_INCREMENT
site_id	bigint(20)		Нет	0	
domain	varchar(200)	utf8_general_ci	Нет		
path	varchar(100)	utf8_general_ci	Нет		
registered	datetime		Нет	0000-00-00 00:00:00	
last_updated	datetime		Нет	0000-00-00 00:00:00	
public	tinyint(2)		Нет	1	
archived	Enum('0', '1')	utf8_general_ci	Нет	0	
mature	tinyint(2)		Нет	0	
spam	tinyint(2)		Нет	0	
deleted	tinyint(2)		Нет	0	
lang_id	int(11)		Нет	0	

wp_blog_versions

В таблице *wp_blog_versions* содержится информация о схеме данных, используемой каждым сайтом сети. Структура таблицы *wp_blog_versions* показана в табл. 12.4.

Таблица 12.4. Структура данных таблицы *wp_blog_versions*

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
blog_id	bigint(20)		Нет	0	
db_version	varchar(20)	utf8_general_ci	Нет		
last_updated	datetime		Нет	0000-00-00 00:00:00	

wp_signups

В таблице *wp_signups* содержится информация обо всех зарегистрированных в сети пользователях. Структура таблицы *wp_signups* показана в табл. 12.5.

Таблица 12.5. Структура данных таблицы wp_signups

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
domain	varchar(200)	utf8_general_ci	Нет		
path	varchar(100)	utf8_general_ci	Нет		
title	longtext	utf8_general_ci	Нет	Нет	
user_login	varchar(60)	utf8_general_ci	Нет		
user_email	varchar(100)	utf8_general_ci	Нет		
registered	datetime		Нет	0000-00-00 00:00:00	
activated	datetime		Нет	0000-00-00 00:00:00	
active	tinyint(1)		Нет	0	
activation_key	varchar(50)	utf8_general_ci	Нет		
meta	longtext	utf8_general_ci	Да	NULL	

wp_registration_log

В таблице wp_registration_log содержится информация обо всех зарегистрированных в сети пользователях, включая идентификатор пользователя, адрес электронной почты, IP-адрес и идентификатор блога. Структура таблицы wp_registration_log показана в табл. 12.6.

Таблица 12.6. Структура данных таблицы wp_registration_log table

Столбец	Тип данных	Схема сопоставления	Нулевое значение	Значение по умолчанию	Дополнительно
ID	bigint(20)		Нет	Нет	AUTO_INCREMENT
email	varchar(255)	utf8_general_ci	Нет		
IP	varchar(30)	utf8_general_ci	Нет		
blog_id	bigint(20)		Нет	0	
date_registered	datetime		Нет	0000-00-00 00:00:00	

Индивидуальные таблицы сайтов

Каждый новый сайт, добавляемый вами в свою сеть, автоматически снабжается идентификатором blog_id. При этом также создается ряд собственных таблиц сайта,

в именах которых присутствует его идентификатор `blog_id`. Допустим, что мы создаем в своей сети первый дополнительный сайт помимо основного сайта. Идентификатор `blog_id` этого сайта будет равен 2, и в базе данных будут созданы следующие таблицы:

◆ <code>wp_\$blog_id_options</code>	◆ <code>wp_\$blog_id_links</code>
◆ <code>wp_\$blog_id_posts</code>	◆ <code>wp_\$blog_id_term_taxonomy</code>
◆ <code>wp_\$blog_id_postmeta</code>	◆ <code>wp_\$blog_id_terms</code>
◆ <code>wp_\$blog_id_comments</code>	◆ <code>wp_\$blog_id_term_relationships</code>
◆ <code>wp_\$blog_id_commentmeta</code>	

Как видите, это такой же набор таблиц, как и в случае стандартного экземпляра WordPress, лишь с тем отличием, что в имени каждой таблицы есть идентификатор `blog_id`. При добавлении в сеть каждого последующего сайта будет создаваться такой же набор таблиц, но с идентификатором `blog_id` нового сайта.



Хотя в WordPress версии 3.5 из панели администратора был удален пользовательский интерфейс для управления ссылками, соответствующая таблица была оставлена для обеспечения обратной совместимости. Если вам нужно управлять ссылками, вы можете воспользоваться плагином [Link Manager \(oreil.ly/FcL3V\)](http://oreil.ly/FcL3V).

Совместно используемые таблицы сайтов

Для всех пользователей многосайтовой сети предусмотрены общие таблицы `wp_users` и `wp_usermeta`.

Привязка пользователей к отдельным сайтам сети осуществляется с помощью нескольких метаключей, размещаемых в таблице `wp_usermeta`. Так, если мы добавим нового пользователя на второй сайт нашей сети, будут созданы следующие метаключи: `primary_blog`, `wp_2_capabilities`, `wp_2_user_level`.

У пользователя может быть только один базовый сайт `primary_blog`, но он может быть привязан к нескольким сайтам с помощью метаключей `capabilities` (полномочия) и `user_level` (уровень пользователя). В случае дефолтного экземпляра WordPress и сайта верхнего уровня многосайтовой сети эти метаключи сохраняются как `wp_capabilities` и `wp_user_level`. При каждом добавлении пользователей на сайты сети создаются новые записи с метаключей для того идентификатора `blog_id`, к которому привязывается пользователь, с назначаемой пользователю ролью.

Сопоставление доменов

По умолчанию любой подсайт многосайтовой сети занимает домен верхнего уровня в составе имени поддомена или подкаталога, если вы специально не дадите ему указание использовать другой домен.

Если вы правильно задали параметры DNS-сервера, сопоставили домен с хостом и определили для хоста запись домена, указывающую на ваш многосайтовый экземпляр, то определить отображение доменов будет совсем несложно:

1. Перейдите в панель администратора сети и выберите в меню пункт **Sites** (Сайты).
2. Наведите указатель мыши на любой подсайт сети. В появившемся меню выберите команду **Edit** (Изменить).
3. Укажите в поле **Site address** (Адрес сайта) то доменное имя, которое вы хотите использовать, проследив за тем, чтобы это была полная форма адреса: **https://<whatever>.com**. Если у вас имеется SSL-сертификат для нового домена, не забудьте его установить и настроить.
4. Подтвердите изменения, щелкнув по кнопке **Save** (Сохранить), и убедитесь, что все работает, как надо, перейдя на новый доменный адрес в браузере.

Если у вас возникнут проблемы с cookie-файлами при попытке войти в серверную часть нового домена, нужно будет модифицировать файл `wp-config.php`, добавив в него следующую строку кода после уже добавленного кода для управления многосайтовостью:

```
define('COOKIE_DOMAIN', $_SERVER['HTTP_HOST']);
```

Некоторые сервисы управляемого хостинга для WordPress-сайтов, как, например, WP Engine, предлагают API, позволяющий программно привязывать домены к нужному DNS-серверу. Это избавляет вас от необходимости тратить время на ручное добавление записей для *дополнительных* или *припаркованных доменов* на вашем хосте. При реализации многосайтового режима WordPress в сочетании с таким API вам потребуется лишь добавить новый подсайт или домен в панели администратора сети WordPress, после чего API произведет всю остальную настройку DNS-записей на хосте. Более подробные сведения можно найти в документации по WP Engine API (wpengineapi.com).



До выхода версии WordPress 4.5 для отображения доменов требовалось устанавливать и настраивать специальный сторонний плагин. Теперь данная возможность входит в состав ядра WordPress.

Некоторые полезные плагины для многосайтового режима

При правильном создании обычного WordPress-плагина он будет работать должным образом и на одном сайте, и в многосайтовой сети. В то же время иногда WordPress-плагины создаются специально для многосайтового режима. Далее представлено краткое описание некоторых наиболее популярных плагинов для многосайтовой сети.

Расширение User Registration для плагина Gravity Forms

Данное расширение (oreil.ly/T9hWN) позволяет вам легко предоставить пользователям возможность регистрации на отдельных сайтах многосайтовой сети, используя для этого короткий код плагина Gravity Forms или произвольно размещаемый виджет. Его также можно настроить на создание в сети нового сайта при создании пользователем своего аккаунта. Это будет полезно в том случае, если вам нужно, чтобы для каждого нового пользователя многосайтовой сети создавался отдельный сайт.

Расширение Member Network Sites для плагина Paid Memberships Pro

Данное расширение для плагина Paid Memberships Pro (oreil.ly/bXtpI) позволяет вам взимать с пользователей плату за создание новых сайтов в вашей сети. О том, как осуществляется взимание платы, мы поговорим в *главе 15*.

Плагин More Privacy Options

После установки данного плагина (bwawwp.com/wp-privacy) в настройках чтения появляется несколько дополнительных уровней конфиденциальности: Network Users Only (Только пользователи сети), Blog Members Only (Только члены блога) и Admins Only (Только администраторы). Эти уровни делают сайт доступным только для соответствующей категории пользователей.

Плагин Multisite Global Media

Плагин Multisite Global Media (oreil.ly/l-Zv0) обеспечивает совместное использование глобальной медиабιβотеки всеми сайтами сети. На самом деле он просто формирует медиабιβотеку некоторого конкретного сайта таким образом, чтобы любой сайт сети мог получить доступ к общим ресурсам.

Плагин Multisite Plugin Manager

Если в своей сети вы хотите предоставить администраторам сайтов возможность управления некоторыми из используемых ими плагинов, то это можно легко реализовать с помощью плагина Multisite Plugin Manager (oreil.ly/gBMbp). Данный плагин предоставляет пользовательский интерфейс администратора сети для задания общесайтовых дефолтных настроек на каждом сайте. При этом вы также можете переопределить эти общесайтовые настройки на определенных сайтах. Имеются три основных режима управления плагинами:

- ◆ Автоматическая активация.
- ◆ Под управлением пользователя.
- ◆ Массовая активация/деактивация.

Плагин Multisite Global Search

Плагин Multisite Global Search (bwawwp.com/wp-globalsearch) позволяет проводить поиск по всем сайтам сети. В комплекте с ним идет встроенный виджет, позволяющий отображать строку поиска в боковой панели. И у виджета, и у страницы результатов имеется настраиваемая таблица стилей. Данный плагин использует короткие коды, позволяющие встроить поиск в любой нужный вам шаблон.

Плагин Multisite Robots.txt Manager

Плагин Multisite Robots.txt Manager (bwawwp.com/wp-robotstxt) позволяет вам создать кастомные файлы robots.txt для каждого сайта сети, а затем быстро их опубликовать в сети или на сайте. Он также обеспечивает мгновенное добавление URL-адресов карты сайта в каждый из файлов robots.txt. Кроме того, данный плагин автоматически выявляет ошибки с кодом 404 и устаревшие файлы robots.txt и позволяет легко исправить их после выявления.

Плагин NS Cloner: Site Copier

Хотя существуют и другие плагины для копирования сайтов в сети, наиболее удачным из них является плагин NS Cloner: Site Copier (oreil.ly/0_k2b). Этот плагин будет очень полезен в том случае, когда в сети требуется копировать настройки и контент с одного сайта на другой. При создании множества однотипных сайтов вы можете создать полностью настроенный "шаблонный" сайт (с нужными настройками тем и плагинов, с дефолтным контентом и т. д.). Вы можете разместить в контенте шаблонного сайта метки-заполнители, чтобы при создании нового сайта они автоматически заменялись на нужные вам значения. Зачем начинать с пустой заготовки сайта, где вам придется все перенастраивать? Просто склонируйте любой нужный вам сайт и возьмите его в качестве отправной точки. Экономьте свое время и деньги!

Плагин WP Multi Network

Плагин WP Multi Network (oreil.ly/PqUxF) позволяет настроить экземпляр WordPress, имеющий несколько сетей с разными наборами подсайтов. Создайте сеть сетей! Если вы помните, при активации многосайтового режима в таблицу `wp_site` вставляется всего одна строка. Плагин WP Multi Network, фактически, позволяет вам вносить в эту таблицу дополнительные строки и создавать подсайты с отличающимися идентификаторами `site_id`.

Основная функциональность многосайтовости

После активации многосайтового режима WordPress у вас появляются функции для работы с многосайтовостью, которые до этого "дремали" в ядре WordPress, ожидая своего часа.

Переменная *\$blog_id*

Поскольку мы уже выяснили, какие таблицы создаются для многосайтовой сети, мы знаем, что каждый сайт сети снабжается уникальным идентификатором `blog_id`. С помощью этого идентификатора вы можете давать системе WordPress указание, на каком сайте следует извлекать или добавлять данные.

Глобальной переменной `$blog_id` автоматически присваивается идентификатор того сайта, на котором вы находитесь, если вы не присвоите ей другой идентификатор с помощью функции `switch_to_blog()`. Эта переменная будет полезна при написании кастомных функций для работы с многосайтовостью, как в следующем примере.

Пример

```
function wds_show_blog_id(){
    global $blog_id;
    echo 'current site id: ' . $blog_id;
}
add_action('init', 'wds_show_blog_id');
```

Если вы находитесь на сайте верхнего уровня, т. е. исходном сайте сети, то данная функция выведет идентификатор "1". Если вы находитесь на сайте, который был создан вторым, то будет выведен идентификатор "2".

Функция *is_multisite()*

Эта функция проверяет, активирован ли многосайтовый режим WordPress. При попытке воспользоваться функциями многосайтового режима без его активации, как правило, выводится сообщение об ошибке. Всегда проверяйте, активирован ли многосайтовый режим, перед выполнением кода, связанного с многосайтовостью, как в следующем примере.

Пример

```
function wds_run_multisite_functions(){
    if (is_multisite())
        echo 'Run whatever WordPress Multisite functionality you want!';
}
add_action('init' , 'wds_run_multisite_functions');
```

Функция *get_current_blog_id()*

Эта функция возвращает идентификатор `blog_id` того сайта, на котором вы находитесь в данный момент. Ее определение содержит буквально две строки кода.

Пример

```
// Функция ядра get_current_blog_id
function get_current_blog_id() {
    global $blog_id;
    return absint($blog_id);
}
```

Функция `get_current_blog_id()` определена в файле `wp-includes/load.php`.

Функция ***switch_to_blog(\$new_blog)***

Данная функция осуществляет переключение от текущего сайта к другому, произвольно заданному сайту. Это полезно в том случае, когда вам требуется извлекать посты или другую информацию из других сайтов сети. После выполнения этих действий можно снова переключиться к текущему сайту с помощью функции `restore_current_blog()`. При использовании этой функции не переключаются автоматически загружаемые опции и плагины. Она принимает один обязательный целочисленный параметр `$new_blog`; это идентификатор того сайта, к которому вам нужно переключиться.

Чтобы переключиться от текущего сайта к другому, мы могли бы выполнить следующий код в любой функции плагина или в файле темы:

Пример

```
echo 'current site id: ' . get_current_blog_id() . '<br>';
switch_to_blog(2);
echo 'new current site id: ' . get_current_blog_id();
```

```
//Этот код сгенерирует следующий вывод:
current site id: 1
new current site id: 2
```

Функция `switch_to_blog()` определена в файле `wp-includes/ms-blogs.php`.

Функция ***restore_current_blog()***

Эта функция предназначена для обратного переключения к текущему сайту после вызова функции `switch_to_blog()`. Она не принимает никаких параметров.

Чтобы снова переключиться к текущему сайту, мы могли бы выполнить следующий код.

Пример

```
echo 'current site id: ' . get_current_blog_id() . '<br>';
switch_to_blog(2);
echo 'new current site id: ' . get_current_blog_id() . '<br>';
```

```
restore_current_blog();  
echo 'original site id: ' . get_current_blog_id();  
  
// Этот код сгенерирует следующий вывод:  
current site id: 1  
new current site id: 2
```

Функция `restore_current_blog()` определена в файле `wp-includes/ms-blogs.php`.

Функция

get_blog_details(\$fields = null, \$get_all = true)

Эта функция извлекает все имеющиеся данные сайта и принимает два параметра:

- ◆ `$fields` — идентификатор или имя конкретного сайта либо массив идентификаторов или имен сайтов. По умолчанию используется идентификатор текущего сайта;
- ◆ `$getall` — логический параметр, по умолчанию равный `true`; указывает, нужно ли включать в возвращаемый объект все доступные данные.

Эта функция возвращает объект, содержащий следующие переменные:

- ◆ `blog_id` — идентификатор сайта, к которому производится обращение;
- ◆ `site_id` — идентификатор сети, к которой привязан данный идентификатор сайта;
- ◆ `domain` — домен, используемый для доступа к сайту;
- ◆ `path` — путь для доступа к сети;
- ◆ `registered` — временная метка момента регистрации сайта;
- ◆ `last_updated` — временная метка последнего обновления сайта;
- ◆ `public` — значение "1" или "0"; указывает, является ли сайт общедоступным;
- ◆ `archived` — значение "1" или "0"; указывает, является ли сайт архивируемым;
- ◆ `mature` — значение "1" или "0"; указывает, содержит ли сайт материалы, рассчитанные на взрослую аудиторию;
- ◆ `spam` — значение "1" или "0"; указывает, был ли сайт помечен как спам;
- ◆ `deleted` — значение "1" или "0"; указывает, был ли сайт удален;
- ◆ `lang_id` — идентификатор языка, на котором написано содержимое сайта;
- ◆ `blogname` — название сайта;
- ◆ `siteurl` — URL-адрес сети, к которой относится сайт;
- ◆ `post_content` — количество постов на сайте.

Чтобы полностью отобразить возвращаемый этой функцией объект, мы могли бы выполнить код следующего примера.

```
$details = get_blog_details(1);
echo '<pre>';
print_r($details);
echo '</pre>';
echo 'Site URL:' . $details->siteurl;
echo 'Post Count:' . $details->post_count;

// Этот код возвратит объект следующего вида:
```

```
stdClass Object
(
    [blog_id] => 1
    [site_id] => 1
    [domain] => schoolpress.me
    [path] => /
    [registered] => 2013-03-01 00:23:26
    [last_updated] => 2013-04-01 14:18:59
    [public] => 1
    [archived] => 0
    [mature] => 0
    [spam] => 0
    [deleted] => 0
    [lang_id] => 0
    [blogname] => School Press
    [siteurl] => http://schoolpress.me
    [post_count] => 10
)
```

Также будет возвращен URL-адрес сети — <http://schoolpress.me>, и количество постов на сайте — 10.

Функция `get_blog_details()` определена в файле `wp-includes/ms-blogs.php`.

Функция

update_blog_details(\$blog_id, \$details = array())

Эта функция обновляет данные сайта и принимает два параметра:

- ◆ `$blog_id` — обязательный целочисленный параметр; идентификатор сайта, данные которого нужно обновить;
- ◆ `$details` — обязательный параметр; массив пар "ключ–значение", где ключи представляют собой любые поля из таблицы сайта, а значения — присваиваемые этим ключам данные.

Чтобы пометить определенный сайт как удаленный, мы могли бы выполнить следующую строчку кода:

```
update_blog_details(2, array('deleted' => '0'));
```

Функция `update_blog_details()` определена в файле `wp-includes/ms-blogs.php`.

Функция ***get_blog_status(\$id, \$pref)***

Эта функция аналогична `get_blog_details()`, но, в отличие от нее, возвращает не объект со всеми полями таблицы `wp_blogs`, а значение некоторого конкретного поля:

- ◆ `$id` — обязательный целочисленный параметр; идентификатор сайта, на котором нужно извлечь значение поля таблицы `wp_blogs`;
- ◆ `$pref` — обязательный строковый параметр; имя поля таблицы `wp_blogs`.

Чтобы отобразить данные о моменте регистрации текущего сайта, мы могли бы выполнить следующую строчку кода:

```
echo get_blog_status(get_current_blog_id(), 'registered');
```

Функция `get_blog_status()` определена в файле `wp-includes/ms-blogs.php`.

Функция ***update_blog_status(\$blog_id, \$pref, \$value)***

Эта функция аналогична `update_blog_details()`, но, в отличие от нее, обновляет не массив полей таблицы `wp_blogs`, а некоторое конкретное поле:

- ◆ `$blog_id` — обязательный целочисленный параметр; идентификатор сайта, на котором нужно обновить поле таблицы `wp_blogs`;
- ◆ `$pref` — обязательный строковый параметр; имя обновляемого поля таблицы `wp_blogs`;
- ◆ `$value` — обязательный строковый параметр; присваиваемое полю значение.

Чтобы пометить текущий сайт как удаленный, мы могли бы выполнить следующую строчку кода:

```
update_blog_status(get_current_blog_id(), 'deleted', '1');
```

Функция `update_blog_status()` определена в файле `wp-includes/ms-blogs.php`.

Функция ***get_blog_option(\$id, \$option, \$default = false)***

Эта функция избавляет вас от необходимости переключаться к сайту с помощью функции `switch_to_blog()`, а затем использовать обычную функцию WordPress `get_option()` или писать кастомный SQL-запрос в том случае, когда требуется получить значение некоторой опции на определенном сайте. Эта функция позволяет получить значение опции на любом сайте сети и принимает следующие параметры:

- ◆ `$id` — обязательный целочисленный параметр; идентификатор сайта, на котором нужно получить значение опции. Можно передать значение `null`, если требуется получить значение опции на текущем сайте;

- ◆ `$option` — обязательный строковый параметр; имя опции, значение которой нужно получить;
- ◆ `$default` — опциональный строковый параметр; результат, возвращаемый в том случае, когда функция не находит указанную опцию.

Чтобы получить значение опции `admin_email` на определенном сайте, мы могли бы выполнить следующий код:

```
echo 'The admin email for site id 2 is ' . get_blog_option(2, 'admin_email');
```

Функция `get_blog_option()` определена в файле `wp-includes/ms-blogs.php`.

Функция ***update_blog_option(\$id, \$option, \$value)***

Эта функция позволяет обновить значение любой опции на конкретном сайте и принимает три параметра:

- ◆ `$id` — обязательный целочисленный параметр; идентификатор сайта, на котором нужно обновить значение опции;
- ◆ `$option` — обязательный строковый параметр; имя опции, значение которой нужно обновить;
- ◆ `$value` — обязательный строковый параметр; значение обновляемой опции.

Чтобы обновить значение опции `admin_email` на определенном сайте, мы могли бы выполнить следующую строчку кода:

```
update_blog_option(2, 'admin_email', 'brian@alphaweb.com');
```

Функция `update_blog_option()` определена в файле `wp-includes/ms-blogs.php`.

Функция ***delete_blog_option(\$id, \$option)***

Эта функция позволяет удалить любую опцию на конкретном сайте и принимает два параметра:

- ◆ `$id` — обязательный целочисленный параметр; идентификатор сайта, на котором нужно удалить опцию;
- ◆ `$option` — обязательный строковый параметр; имя удаляемой опции.

Чтобы удалить кастомную опцию на определенном сайте, мы могли бы выполнить следующую строчку кода:

```
delete_blog_option(2, 'wds_custom_option');
```

Функция `delete_blog_option()` определена в файле `wp-includes/ms-blogs.php`.

Функция ***get_blog_post(\$blog_id, \$post_id)***

Эта функция позволяет получить некоторый пост на любом сайте сети и принимает два параметра:

- ◆ `$blog_id` — обязательный целочисленный параметр; идентификатор сайта, на котором нужно получить пост;

- ◆ `$post_id` — обязательный целочисленный параметр; идентификатор извлекаемого поста.

Чтобы получить заголовок третьего поста на втором сайте нашей сети, мы могли бы выполнить следующий код:

```
$post = get_blog_post(2, 3);  
echo $post->post_title;
```

Функция `get_blog_post()` определена в файле `wp-includes/ms-functions.php`.

Функция

`add_user_to_blog($blog_id, $user_id, $role)`

Эта функция позволяет добавить пользователя на любом сайте сети, назначив ему определенную роль. Она принимает три параметра:

- ◆ `$blog_id` — обязательный целочисленный параметр; идентификатор сайта, на котором нужно добавить пользователя;
- ◆ `$user_id` — обязательный целочисленный параметр; идентификатор добавляемого пользователя;
- ◆ `$role` — обязательный строковый параметр; назначаемая пользователю роль.

В случае успешного добавления пользователя функция возвращает значение `true`; в противном случае возвращается объект `WP_Error`.

Чтобы добавить определенного пользователя на втором сайте нашей сети, назначив ему роль администратора, мы могли бы выполнить следующую строчку кода:

```
add_user_to_blog(2, 5, 'administrator');
```

Функция `add_user_to_blog()` определена в файле `wp-includes/ms-functions.php`.

Функция **`wpmu_delete_user($user_id)`**

Эта функция удаляет из всей сети указанного пользователя и все посты, автором которых он является.

- ◆ `$user_id` — обязательный целочисленный параметр; идентификатор удаляемого пользователя.

В случае успешного удаления пользователя возвращается значение `true`; в противном случае возвращается значение `false`.

Когда активирован многосайтовый режим, обычная функция `wp_delete_user()` удаляет пользователя только на текущем сайте, не удаляя его полностью. Поэтому важно проследить за тем, чтобы в многосайтовом режиме использовалась именно функция `wpmu_delete_user()`, с одновременным подключением к хукам `wp_delete_user` и `wpmu_delete_user`. Если же вам нужно выполнять код и в многосайтовом, и в односайтовом режиме, то это можно делать применяя функцию `is_multisite()`, как показано в следующем примере.

```
if (is_multisite()) {
    wpmpu_delete_user($user_id);
} else {
    wp_delete_user($user_id);
}
```

Функция `wpmpu_delete_user()` определена в файле `wp-includes/ms-functions.php`.

Функция *create_empty_blog* (*\$domain*, *\$path*, *\$weblog_title*, *\$site_id* = 1)

Эта функция создает в сети новый сайт, предварительно убедившись, что такой сайт еще не существует. Она может использоваться администратором сети для добавления новых сайтов и принимает четыре параметра:

- ◆ *\$domain* — обязательный строковый параметр; домен нового сайта;
- ◆ *\$path* — обязательный строковый параметр; путь нового сайта;
- ◆ *\$weblog_title* — обязательный строковый параметр; заголовок или название нового сайта;
- ◆ *\$site_id* — опциональный целочисленный параметр; идентификатор сети, к которой привязывается новый сайт. По умолчанию задан идентификатор 1.

Чтобы добавить новый сайт в нашу сеть, мы могли бы выполнить следующий код:

```
create_empty_blog('someteacher.schoolpress.me', '/', 'Mr. Some Teacher');
```

Функция `create_empty_blog()` определена в файле `wp-includes/ms-functions.php`.

Не упомянутые здесь функции

Мы не стали рассматривать здесь буквально все функции для работы с многосайтовостью, упомянув лишь наиболее важные из них. В зависимости от того, что вы пытаетесь создать, вам могут потребоваться и некоторые другие функции. Чтобы ознакомиться со всеми имеющимися функциями для работы с многосайтовостью в WordPress, обратитесь к исходному коду. Определение этих функций содержится в следующих файлах:

- ◆ `wp-admin/includes/ms.php`;
- ◆ `wp-includes/ms-blogs.php`;
- ◆ `wp-includes/ms-functions.php`.

Более подробные сведения можно найти в разделе "Функции многосайтового режима" Кодекса WordPress (oreil.ly/FMq2b).

Локализация приложений WordPress

Локализацией (или *интернационализацией*) называют процесс перевода приложения для использования различных локалей и языков. В этой главе мы рассмотрим доступные вам инструменты и методы для локализации приложений, тем и плагинов.



В англоязычной литературе иногда встречаются сокращения "l10n" (localization) и i18n (internationalization).

Нужна ли локализация вашему приложению?

Рынок веб-приложений с каждым днем приобретает все более глобальный характер. Обеспечив поддержку нескольких языков в своем приложении, вы получите серьезное преимущество на рынке по сравнению с конкурентами, использующими только вашу основную локаль/язык или только другие локали/языки.

Если вы собираетесь выпустить некоторую программу под лицензией с открытым исходным кодом, то, локализовав ее перед выпуском, вы существенно расширите круг разработчиков, способных принять участие в вашем проекте. Выпустив локализованный плагин или тему, вы дадите возможность разработчикам, говорящим на других языках, вносить вклад в развитие проекта напрямую, вместо того чтобы создавать отдельное ответвление для обеспечения поддержки других языков.

Если вы собираетесь распространять коммерческий плагин или тему, то локализация кода позволит вам увеличить число потенциальных покупателей.

Если ваш целевой рынок ограничивается Соединенными Штатами и вы пока не планируете расширяться на другие регионы или языки, то, возможно, вам не стоит тратить время на подготовку кода к локализации. Также следует иметь в виду, что каждая дополнительная версия приложения с поддержкой нового языка или региона, как правило, требует дополнительных затрат на хостинг, справочную систему, службу поддержки и техническое обслуживание. Многие компании не могут позволить себе эти затраты на ранней стадии жизненного цикла приложения. В то же время, как вы увидите далее, базовый процесс подготовки кода к локализации (обертывание строкового вывода в функцию `_()`, `_e()`, или `_x()`) не представляет больших сложностей и часто несет дополнительную пользу помимо локализации.

Наконец, важно отметить, что локализация часто подразумевает не только перевод кода на другой язык. Если ваш код взаимодействует с другими сервисами, то вам следует убедиться, что эти сервисы доступны в других регионах, и, если это не так, разработать альтернативные решения. Например, важной составляющей плагина Paid Memberships Pro является интеграция с платежными системами. Поэтому перед выполнением локализации Джейсон убедился в том, что этот плагин хорошо интегрируется с международными платежными системами. Если бы это было не так, то пользователям был бы доступен плагин на соответствующем языке, но он не смог бы взаимодействовать с платежной системой, действующей в их регионе.

Как выполняется локализация в WordPress

WordPress использует систему перевода *gettext*, разработанную в рамках проекта GNU. В WordPress эта система содержит следующие компоненты:

- ◆ способ определения локали/языка;
- ◆ способ определения текстовых доменов;
- ◆ способ перевода строк в коде;
- ◆ файлы *.pot, содержащие все переводимые слова и фразы;
- ◆ файлы *.po для каждого языка, содержащие переводы;
- ◆ файлы *.mo для каждого языка, содержащие скомпилированную версию переводов из файлов *.po.

Чтобы ваши переводы работали, должны присутствовать все эти компоненты. В следующих далее разделах мы поочередно рассмотрим все выполняемые шаги. После их выполнения у нас будут все необходимые инструменты для создания локализованного плагина и переведенных файлов локали.

Определение локали в WordPress

Чтобы определить локаль в WordPress, просто задайте константу `WPLANG` в файлах `wp-config.php`:

```
<?php
// Подключаем локаль "испанский язык/Испания" и языковые файлы
define('WPLANG', 'es_ES');
?>
```



Термин "локаль" вместо термина "язык" обусловлен тем, что у одного и того же языка может быть несколько вариантов перевода. В качестве примера можно вспомнить, что в Великобритании и США существуют разные варианты английского языка, а в Мексике и Испании — разные варианты испанского языка.

Текстовые домены

Для организации таблиц с переводами библиотека gettext использует *текстовые домены*. В случае WordPress это выражается в том, что каждый плагин и тема должны иметь собственный уникальный текстовый домен.

С технической точки зрения, в качестве текстовых доменов возможны любые уникальные и единообразные строки с надлежащим синтаксисом (из строчных символов, с дефисами, но без символов подчеркивания). Однако на практике текстовый домен для плагина или темы должен совпадать со слагом, поскольку большинство других плагинов и инструментов будут рассчитывать на это. Например, плагины из репозитория WordPress.org требуют, чтобы текстовый домен совпадал со слагом плагина, поскольку это нужно для надлежащей работы плагина GlotPress и других инструментов, представленных на этом сайте.

Вот несколько примеров того, какие текстовые домены могут встречаться в реальных проектах:

- ♦ для всего кода ядра WordPress предусмотрен текстовый домен `default`;
- ♦ плагин Paid Memberships Pro использует текстовый домен `paid-memberships-pro`;
- ♦ тема Memberlite задействует текстовый домен `memberlite`.

Настройка текстового домена

Система WordPress должна знать, где расположены файлы локализации каждого локализуемого плагина или темы вашего сайта. Предоставить ей эту информацию можно с помощью функции `load_plugin_textdomain()`, `load_textdomain()` или `load_theme_textdomain()`. Эти три функции различаются лишь тем, что принимают разные параметры и предназначены для разных ситуаций.

Какую бы функцию вы ни выбрали, ее следует вызывать в приложении как можно раньше, поскольку любые строки, обрабатываемые или выводимые с помощью функций перевода, будут переводиться только после загрузки текстового домена.

Далее показано, как можно реализовать каждый из трех способов загрузки текстового домена в файле `includes/localization.php`:

- ♦ `load_plugin_textdomain($domain, $abs_rel_path, $plugin_rel_path)` — эта функция принимает три параметра. В качестве первого параметра `$domain` передается домен вашего плагина или приложения (в нашем случае `schoolpress`). В дополнение к первому параметру нужно предоставить второй или третий параметр, указывающий расположение языковой папки, из которой будет загружаться файл `*.mo`. Параметр `$abs_rel_path` уже является устаревшим, но пока остается доступным для обеспечения обратной совместимости. Просто передайте в качестве него значение `false` и воспользуйтесь параметром `$plugin_rel_path`, как показано в следующем примере.

Пример

```
<?php
function schoolpress_load_textdomain(){
    //Загружаем текстовый домен из папки /plugins/schoolpress/languages/
    load_plugin_textdomain(
        'schoolpress',
        FALSE,
        dirname(plugin_basename(__FILE__)) . '/languages/'
    );
}
add_action('init', 'schoolpress_load_textdomain', 1);
?>
```

Этот код загружает нужный языковой файл из языковой папки в зависимости от того, как будет задана константа `WPLANG` в файле `wp-config.php`. Получив путь к текущему файлу с помощью функции `plugin_basename(__FILE__)`, мы применяем к нему функцию `dirname()`, чтобы получить путь к корневой папке плагина, поскольку мы находимся в подпапке `includes` внутри папки `schoolpress` нашего плагина.



Вы также можете увидеть определение текстового домена в заголовке PHP-комментария плагина или темы. Эта настройка используется системой WordPress с целью обеспечить перевод метаинформации плагина даже в том случае, когда сам плагин отключен. Начиная с версии WordPress 4.6, такая настройка текстового домена в заголовке является необязательной; при этом по умолчанию используется слаг плагина или темы.

- ◆ `load_theme_textdomain($domain, $path)` — функция позволяет загрузить языковые файлы (при их наличии), специально предназначенные для вашей темы, как показано в следующем примере.

Пример

```
<?php
function schoolpress_load_textdomain() {
    load_theme_textdomain(
        'schoolpress', get_template_directory() . '/languages/'
    );
}
add_action('init', 'schoolpress_load_textdomain', 1);
?>
```

- ◆ `load_textdomain($domain, $path)` — функция обеспечивает возможность загрузить текстовый домен, но при этом придется получить настройку локали самостоятельно.

Хотя в случае плагинов и тем прямой вызов функции `load_textdomain()` не рекомендуется, этот подход может оказаться полезным в тех ситуациях, когда один и тот же домен требуется для множества разных плагинов. Прямой вызов этой функции также дает вам некоторую гибкость, позволяя предоставить другим разработчикам возможность легкой замены и расширения ваших языковых файлов. Код, приведенный в следующем примере, позволяет сначала загружать файл `*.mo` из глобального языкового каталога WordPress (обычно `wp-content/languages/`), а затем файл `*.mo` из локального языкового каталога вашего плагина. Это дает возможность другим разработчикам переопределять ваши переводы путем добавления собственных файлов `*.mo` в глобальный языковой каталог.

Пример

```
<?php
function schoolpress_load_textdomain() {
    // Получаем локаль
    $locale = apply_filters('plugin_locale',
                           get_locale(), 'schoolpress');
    $mofile = 'schoolpress-' . $locale . '.mo';
/*
Пути к локальному и глобальному языковым файлам (файлам плагина и WP).
Примечание: при размещении этого кода вне основного файла плагина
нужно модифицировать вызов функции dirname(__FILE__).
*/

    $mofile_local  = dirname(__FILE__).'/languages/' . $mofile;
    $mofile_global = WP_LANG_DIR . '/schoolpress/' . $mofile;

    // сначала загружаем глобальный файл
    load_textdomain('schoolpress', $mofile_global);

    // затем загружаем локальный файл
    load_textdomain('schoolpress', $mofile_local);
}
add_action('init', 'schoolpress_load_textdomain', 1);
?>
```

В данном примере мы получаем локаль с помощью функции `get_locale()` и применяем фильтр `plugin_locale`, после чего ищем файл `*.mo` в глобальном языковом каталоге (обычно `/wp-content/languages/`) и языковом каталоге нашего плагина.

Подготовка строк с помощью функций перевода

Первое, что нужно сделать для локализации кода, — обернуть каждую отображаемую строку в одну из функций перевода, предоставляемых системой WordPress.

Все эти функции работают примерно одинаково: им передается некоторый исходный текст вместе с доменом и/или другой информацией, указывающей переводчику, какой контекст следует использовать при переводе этого текста.

Давайте подробнее рассмотрим наиболее распространенные функции.

Функция `__($text, $domain = "default")`

Эта функция принимает два параметра: `$text` — переводимый текст, и `$domain` — домен вашего плагина или темы. Она возвращает текст, переведенный с учетом того, какой домен и язык заданы в файле `wp-config.php`.



Функция `__()` в действительности является псевдонимом функции `translate()`, используемой системой WordPress в фоновом режиме. Хотя ничто не мешает вам вызывать функцию `translate()` напрямую, в силу краткости функции `__()` вы будете использовать ее гораздо чаще.

Вот пример того, как можно обернуть несколько строк с помощью функции `__()`.

Пример

```
<?php
// Присвоение переменной строки без локализации
$title = 'Assignments';

// присвоение переменной строки с локализацией
$title = __('Assignments', 'schoolpress');
?>
```

Функция `_e($text, $domain = "default")`

Эта функция принимает два параметра: `$text` — переводимый текст, и `$domain` — домен вашего плагина или темы. Она выводит текст, переведенный с учетом того, какой домен и язык заданы в файле `wp-config.php`.

Эта функция делает то же самое, что и функция `__()`, но не возвращает полученный результат, а выводит его на экран. Вот пример того, как можно обернуть несколько строк с помощью функции `_e()`.

Пример

```
<?php
// Вывод значения переменной без локализации
?>
<h2><?php echo $title; ?></h2>
<?php
// вывод значения переменной с локализацией
?>
<h2><?php _e($title, 'schoolpress'); ?></h2>
```

На практике вы будете использовать функцию `__()` для присвоения значения переменной, а функцию `_e()` — для вывода значения переменной.

Функция `_x($text, $context, $domain = "default")`

Эта функция принимает три параметра: `$text` — переводимый текст, `$context` — используемый при переводе контекст, и `$domain` — домен вашего плагина или темы. Она возвращает текст, переведенный с учетом того, какой контекст, домен и язык заданы в файле `wp-config.php`.

Функция `_x()` делает то же самое, что и функция `__()`, но дает переводчикам дополнительное указание о том, как должен переводиться текст, с помощью параметра `$context`. Это требуется в тех случаях, когда в коде имеется несколько вхождений одного и того же слова или фразы, которые могут требовать различного перевода.

Так, в английском языке слово *"title"* может означать и название книги, и титул человека, такой как "мистер" или "миссис". В других языках при этом будут использоваться разные слова в зависимости от контекста. Провести различие между этими контекстами можно с помощью функции `_x()`.

Следующий (более сложный) пример показывает, как можно настроить пару переменных для создания класса приложения SchoolPress.

Пример

```
<?php
$class_title_field_label = _x('Title', 'class title', 'schoolpress');
$class_professor_title_field_label = _x('Title', 'name prefix', 'schoolpress');
?>

<h3>Class Description</h3>
<label><?php echo $class_title_field_label; ?></label>
<input type="text" name="title" value="" />

<h3>Professor</h3>
<label><?php echo $class_professor_title_field_label; ?></label>
<input type="text" name="professor_title" value="" />
```



Функции `_x()` и `_ex()` иногда называют "объясняющими" функциями, поскольку используемый в них параметр `$context` дополнительно объясняет, как должен переводиться текст.

Функция `_ex($title, $context, $domain = "default")`

Функция `_ex()` делает то же самое, что и функция `_x()`, но не возвращает переведенный текст, а выводит его на экран.

Сочетание перевода с экранированием

В главе 7 мы говорили о том, насколько важно экранировать строки, отображаемые в HTML-атрибутах или других критически важных областях. Если вам также нужно перевести такие строки, то вместо того, чтобы вызывать две функции, можно воспользоваться одной из функций WordPress для одновременного выполнения этих операций. Эти функции дают точно такой же результат, как в том случае, если бы вы сначала вызвали функцию перевода, а затем — функцию экранирования:

- ◆ `esc_attr__()`
- ◆ `esc_attr_e()`
- ◆ `esc_attr_x()`
- ◆ `esc_html__()`
- ◆ `esc_html_e()`
- ◆ `esc_html_x()`

Создание и загрузка файлов перевода

После того как вы выполните разметку кода для функций перевода, вам потребуется сгенерировать файл *.pot, который будет необходим переводчикам для перевода вашего приложения. Файл *.pot будет содержать секции следующего вида для каждой присутствующей в вашем коде строки:

```
#: schoolpress.php:108
#: schoolpress.php:188
#: pages/courses.php:10
msgid "School"
msgstr ""
```

Код этой секции указывает, что в строках 108 и 188 файла schoolpress.php и в строке 10 файла pages/courses.php используется слово "School".

Чтобы теперь создать испанский перевод плагина, нужно сделать копию файла schoolpress.pot под именем schoolpress-es_ES.po и дополнить в этом файле строки с ключом msgstr. Вот как это будет выглядеть:

```
#: schoolpress.php:108
#: schoolpress.php:188
#: pages/courses.php:10
msgid "School"
msgstr "Escuela"
```

Теперь нужно скомпилировать эти файлы *.po в формат *.mo, оптимизированный для обработки переводов.

В случае больших плагинов и приложений нецелесообразно вручную находить номера строк кода для каждой переводимой строки и изменять эти данные по мере обновления плагина. Далее мы пошагово покажем, как с помощью консольной утилиты xgettext операционной системы Linux можно сгенерировать файл *.pot, а с

помощью утилиты `msgfmt` скомпилировать файлы `*.po` в файлы `*.mo`. Также можно воспользоваться бесплатной программой **Poedit** (www.poedit.net), которая предлагает удобный графический интерфейс для сканирования кода и создания файлов `*.pot`, `*.po` и `*.mo` и доступна для операционных систем Windows, MacOS и Linux.

Организация файлов локализации

Перед тем как погружаться в специфику создания этих файлов, давайте разберемся с тем, как эти файлы обычно сохраняются в плагине. В случае нашего приложения SchoolPress файлы локализации сохраняются в папке `languages` внутри основного плагина приложения. Весь код локализации, включая вызов функции `load_plugin_textdomain()`, размещается в файле `localization.php` внутри каталога `includes`.

Таким образом, наши файлы организованы следующим образом:

- ◆ `../plugins/schoolpress/schoolpress.php` (включая файл `localization.php`)
- ◆ `../plugins/schoolpress/includes/localization.php` (загружает текстовый домен и другие функции локализации)
- ◆ `../plugins/schoolpress/languages/schoolpress.pot` (список переводимых строк)
- ◆ `../plugins/schoolpress/languages/schoolpress.po` (дефолтный/английский перевод)
- ◆ `../plugins/schoolpress/languages/schoolpress.mo` (скомпилированный вариант дефолтного/английского перевода)
- ◆ `../plugins/schoolpress/languages/schoolpress-es_ES.po` (испанский перевод)
- ◆ `../plugins/schoolpress/languages/schoolpress-es_ES.mo` (скомпилированный вариант испанского перевода)

При разработке крупного приложения с множеством кастомных плагинов и тем будет проще локализовать все плагины и темы по отдельности, а не создавать один общий файл перевода. Если плагины планируется использовать только в одном проекте, то их можно встроить в основной плагин приложения в виде модульных файлов `*.php`. Если же плагины могут предназначаться для другого проекта, то их следует локализовать по отдельности, чтобы файлы локализации можно было переносить вместе с плагином.

Генерирование файла `*.pot`

Давайте сформируем файл `*.pot` для нашего плагина, используя утилиту `xgettext`, которая практически всегда присутствует в операционной системе Linux¹.

Чтобы сгенерировать файл `*.pot` для нашего приложения SchoolPress, следует ввести в командной строке команду `cd` для перехода в каталог основного плагина при-

¹ Если эта утилита не установлена в вашей операционной системе, найдите и установите версию пакета `gettext`, предназначенную для вашего дистрибутива Linux.

ложения `wp-content/plugins/schoolpress`, а затем выполнить команды, перечисленные в следующем примере.

Пример

```
xgettext -o languages/schoolpress.pot \  
--default-domain=schoolpress \  
--language=PHP \  
--keyword=_ \  
--keyword=__ \  
--keyword=_e \  
--keyword=_ex \  
--keyword=_x \  
--keyword=_n \  
--sort-by-file \  
--copyright-holder="SchoolPress" \  
--package-name=schoolpress \  
--package-version=1.0 \  
--msgid-bugs-address="info@schoolpress.me" \  
--directory=. \  
$(find . -name "*.php")
```

Давайте разберемся с тем, что делает этот код:

- ♦ `-o languages/schoolpress.pot` — этот параметр указывает расположение выходного файла;
- ♦ `--default-domain=schoolpress` — этот параметр определяет текстовый домен как `schoolpress`;
- ♦ `--language=PHP` — этот параметр сообщает утилите `xgettext` о том, что мы используем PHP-код;
- ♦ `--keyword=...` — этот параметр настраивает утилиту `xgettext` на извлечение всех строк, передаваемых указанной функции. Если помимо указанных здесь функций у вас присутствуют какие-либо дополнительные функции перевода (например, функция `esc_attr__`), то не забудьте добавить параметр `keyword` и для этих функций;
- ♦ `--sort-by-file` — этот параметр обеспечивает сортировку выходных данных по имени файла, если это возможно;
- ♦ `--copyright-holder="SchoolPress"` — этот параметр задает данные о владельце авторских прав, указываемые в заголовке файла `*.pot`. Здесь следует указать лицо или организацию, владеющую авторскими правами на создаваемое приложение, плагин или тему;



Информация с сайта gnu.org (bit.ly/gnu-gettext): "Предполагается, что переводчики откажутся от авторских прав на свои переводы или передадут их другим лицам, чтобы сопровождающие пакеты могли распространять их без рисков правового характера. При отсутствии информации о владельце авторских прав выходные

файлы помечаются как общедоступные; при этом, опять же, предполагается, что переводчики откажутся от своих авторских прав, чтобы сопровождающие пакетов могли распространять их без рисков правового характера".

- ◆ `--package-name=schoolpress` — этот параметр задает имя пакета, указываемое в заголовке файла `*.pot`. Обычно это имя совпадает с доменным именем;
- ◆ `--package-version=1.0` — этот параметр задает номер версии пакета, указываемый в заголовке файла `*.pot`. Этот номер должен обновляться при выпуске каждой новой версии вашего приложения, плагина или темы;
- ◆ `--msgid-bugs-address="info@schoolpress.me"` — этот параметр задает адрес электронной почты, указываемый в заголовке файла `*.pot`. На этот адрес отправляются сообщения о выявленных в файле `*.pot` ошибках;
- ◆ `--directory=.` — этот параметр дает утилите `xgettext` указание начать сканирование с текущего каталога;
- ◆ `$(find . -name "*.php")` — эта команда дает операционной системе Linux указание найти все файлы `*.php`, расположенные в текущем каталоге.

Создание файла `*.po`

Как вы уже знаете, программа Poedit предлагает удобный графический интерфейс для создания файлов `*.po` на основе файлов `*.pot` и ввода перевода каждой переводимой строки. Однако вы без труда можете сделать это и сами: просто скопируйте файл `*.pot` в файл `*.po` (например, `es_ES.po`) в языковом каталоге, а затем отредактируйте файл `*.po`, введя свои переводы в строках с ключом `msgstr`.

Создание файла `*.mo`

После того как вы модифицируете файлы `*.po` под нужную вам локаль, их нужно будет скомпилировать в файлы `*.mo`. Для создания файлов `*.mo` можно использовать программу `msgfmt` для Linux, вызывая ее с помощью команд вида `msgfmt es_ES.po --output-file es_ES.mo`.

GlottPress

GlottPress — это инструмент, позволяющий переводчикам совместно работать над переводами в онлайн-режиме. Вместо управления отдельными файлами `*.po` для каждой локали GlottPress предлагает веб-интерфейс для редактирования произвольной строки под любую локаль. Переводы сохраняются в базе данных, и, когда доля переведенных строк достигает определенного порога, GlottPress автоматически генерирует файлы `*.po` и `*.mo`. Вам даже не потребуется скачивать сгенерированные языковые пакеты и встраивать их в свой плагин. WordPress найдет и скачает их автоматически с учетом выбранной вами локали.

Использование GlotPress для ваших плагинов и тем в репозитории WordPress.org

Если вы разместили свой плагин или тему в репозитории WordPress.org, то для использования GlotPress будет достаточно лишь должным образом обернуть строки для перевода и убедиться, что ваш текстовый домен совпадает со слагом плагина или темы. Если вы выполните эти условия, то для вас будет сгенерирован новый проект перевода, расположенный по одному из следующих адресов:

◆ <https://translate.wordpress.org/projects/wp-plugins<your-plugins-slug>/>

◆ <https://translate.wordpress.org/projects/wp-themes<your-themes-slug>/>

Вот и все! Если вы уже встроили имеющиеся файлы *.po для вашего плагина в папке /languages/, то они будут импортированы в проект GlotPress в виде пакетов соответствующих локалей с уже переведенными строками.

После того как вы интегрируете свой плагин или тему с GlotPress, вам больше не потребуется встраивать языковые файлы в свой проект. WordPress будет автоматически искать языковые пакеты с учетом вашей локали и автоматически скачивать их для использования. Однако обратите внимание, что автоматически распространяются только те пакеты локалей, в которых доля переведенных строк составляет не менее 95%. Поэтому проследите за тем, чтобы ваши переводы были достаточно полными. В качестве альтернативы можно скачать файлы *.po и *.mo неполных переводов и встроить их в свой плагин, как было показано ранее в этой главе.

Создание собственного сервера GlotPress

Преимущества технологии GlotPress можно воспользоваться и при размещении плагина или темы на собственном сервере — для этого потребуется настроить собственный сервер перевода. Разработчики этой технологии создали плагин для WordPress, который можно установить и активировать на любом WordPress-сайте. Плагин GlotPress (oreil.ly/t_p1b) настроит конечную точку с адресом /glotpress/, где вы сможете создавать новые проекты перевода для плагинов и тем, размещенных вами на собственном сервере.

После того как вы настроите свой сервер GlotPress, вы сможете создать новый проект для своего плагина, размещенного не на сайте WordPress.org. WordPress не будет автоматически скачивать эти переводы для ваших пользователей, но вы сможете экспортировать файлы *.po и *.mo для подключения их к плагину.



Еще одна возможность состоит в том, чтобы с помощью хука `translations_api` (oreil.ly/pWJcp) настроить WordPress на скачивание доступных переводов с собственного сервера GlotPress, но мы пока не видели примеров использования этой возможности. Если вас заинтересовал такой подход, попробуйте его реализовать в качестве факультативного задания. На наш взгляд, пока что проще включать файлы перевода в состав распространяемого вами плагина.

В зависимости от выбранного сценария веб-приложения, перевод его на другие языки может сыграть важную роль в обеспечении успеха. При создании кастомной темы или плагина также рекомендуется всегда предусматривать возможность локализации.

Оптимизация и масштабирование WordPress

В данной главе мы поговорим о том, как можно "выдавить" из WordPress максимальную производительность с помощью оптимальной конфигурации сервера, кэширования и хорошо продуманного кода.

Систему WordPress часто упрекают в том, что она не масштабируется столь же хорошо, как другие PHP-фреймворки или языки программирования. Это расхожее мнение о том, что WordPress плохо справляется с ростом масштаба, сформировалось, главным образом, потому, что данная CMS изначально служила для создания небольших блогов на базе совместно используемых аккаунтов хостинга. Кроме того, разработчики ядра WordPress приняли ряд решений (например, решение о поддержке устаревшей функциональности и старых версий PHP и MySQL), для обеспечения поддержки как можно большего числа возможных конфигураций хостинга, включая малопроизводительные совместные аккаунты хостинга.

Поэтому во Всемирной сети действительно присутствует множество очень медленных WordPress-сайтов, подкрепляющих мнение о том, что система WordPress является малопроизводительной сама по себе. Однако *система WordPress работает чертовски быстро при хорошей конфигурации, и к ней могут применяться такие же методы масштабирования, как и к любому другому приложению на базе PHP и MySQL*. В этой главе мы рассмотрим многие из методов и познакомимся с рядом инструментов и концепций, которые могут оказаться полезными при разработке WordPress-приложений.

Терминология

В этой главе и на протяжении всей книги часто встречаются слова "оптимизация" и "масштабирование". Важно четко понимать смысл этих понятий.

Оптимизация — под "оптимизацией" обычно понимается обеспечение максимальной быстрой работы приложения и скриптов. Иногда требуется оптимизировать не скорость работы, а потребление памяти или какой-либо иной параметр. Однако в большинстве случаев "оптимизация" подразумевает ускорение различных операций.

Масштабирование — этот термин подразумевает создание приложения, способного справляться со все большим числом различных вещей — с большим числом про-

смотров страниц, посещений, пользователей, сообщений, файлов, подсайтов, вычислений.

Под "масштабированием" также часто понимается создание приложения, способного справляться с *более крупными вещами* — более крупными страницами, постами, файлами.

Следует отметить, что иногда приложение или определенные части приложения прекрасно работают при малой нагрузке или, скажем, небольшом размере таблиц базы данных. Но как только возрастает число пользователей и число или размеры обрабатываемых объектов, приложение существенно замедляется или полностью отказывается работать.

Под "масштабируемостью" понимается субъективная мера того, насколько хорошо ваш код или приложение справляются с большим числом различных вещей или вещами большего размера. В большинстве случаев следует создавать приложение с расчетом на ожидаемый вами рост нагрузки, предусмотрев при этом некоторый запас. В то же время необходимо всегда оценивать плюсы и минусы каждого решения в отношении платформы или кода, принимаемого вами с целью повышения масштабируемости. Каждое такое решение обычно влечет определенные издержки как в денежном выражении, так и в виде технической задолженности или повышения сложности кодовой базы. Кроме того, некоторые методы предельно ускоряют обработку большого числа крупных транзакций, но фактически снижают скорость, когда приходится иметь дело с небольшим числом небольших транзакций. Поэтому всегда важно создавать приложение с учетом реально ожидаемой нагрузки и не писать повышающий масштабируемость код без особой необходимости.

Масштабирование и оптимизация тесно связаны друг с другом, поскольку быстрые приложения легче поддаются масштабированию. Хотя масштабирование включает в себя не только использование быстродействующих компонентов, быстрые компоненты существенно упрощают этот процесс, а медленное приложение, наоборот, может сделать его затруднительным. По этой причине оптимизировать приложение всегда лучше в направлении изнутри наружу. В отлично написанной белой книге "Правда о производительности в WordPress" от компаний Copyblogger Media и W3 Edge (oreil.ly/8ajHj) проводится сравнение оптимизации "источника" и оптимизации "внешнего окружения":

Источник — под "источником" понимается ваше WordPress-приложение как источник всех поступающих из него данных. Оптимизация источника подразумевает ускорение вашего WordPress-приложения и используемого им сервера.

Внешнее окружение — под "внешним окружением" понимаются внешние сервисы, которые больше удалены от источника, но часто расположены ближе к пользователям. К таким сервисам относятся, например, сети доставки контента (CDN) и такие операции, как кэширование браузера. Оптимизация внешнего окружения подразумевает продуманное использование этих сервисов для улучшения пользовательского интерфейса конечных пользователей.

Источник или внешнее окружение?

Как уже упоминалось, проводить оптимизацию лучше в направлении изнутри наружу, т. е. от источника к внешнему окружению. Повышение производительности ядра WordPress всегда "просочится" через внешнее окружение до конечных пользователей. С другой стороны, хотя повышение производительности внешних сервисов может улучшить пользовательский опыт, при этом часто упускаются из виду серьезные нерешенные проблемы в источнике.

Типичный пример такой ситуации — использование прокси-сервера наподобие Varnish (о котором мы поговорим подробнее далее в этой главе) для сокращения времени загрузки на медленно загружающемся сайте. Varnish делает копии полностью отрендеренных страниц WordPress-сайта. Если затем посетитель запросит страницу, копия которой присутствует в кэше сервера Varnish, то ему будет возвращена эта копия, вместо того чтобы генерировать эту страницу заново с помощью WordPress.

Поскольку доставка статичных файлов занимает гораздо меньше времени, чем выполнение динамического PHP-кода, Varnish может обеспечить существенное ускорение сайта. Страница, загружаемая за 10 секунд в медленной конфигурации WordPress, может загружаться лишь за 1 секунду при доставке ее копии с помощью Varnish. Однако в некоторых случаях время загрузки будет по-прежнему составлять неприемлемо большой промежуток в 10 секунд. Так, оно будет равно 10 секундам при первой загрузке страницы. Страницы в панели администратора также будут загружаться за 10 секунд. Если копия определенной страницы будет удалена из кэша сервера Varnish, например потому, что ее содержимое изменится, или потому, что серверу Varnish потребуется высвободить пространство, то на загрузку свежей копии этой страницы, опять же, уйдет 10 секунд.

Varnish и другие аналогичные инструменты обеспечивают потрясающие результаты и могут стать ценным элементом платформы вашего приложения. В то же время важно убедиться в том, что такие внешние сервисы не скрывают за собой какие-либо проблемы в источнике.



Будьте осмотрительны: значительная часть доступных в Интернете советов по ускорению сайтов рассчитана на статические сайты, а здесь идет речь о создании веб-приложений с некоторыми интерактивными компонентами. В то же время, хорошая система кэширования снижает нагрузку на веб-серверы, высвобождая ресурсы центрального процессора и памяти, что может улучшить производительность некешируемых страниц.

Тестирование

В данной главе производительность понимается, главным образом, как скорость загрузки определенных страниц в браузере. С помощью нескольких разных инструментов мы будем оценивать время загрузки страницы при ее открытии одним пользователем и при ее открытии сразу большим количеством пользователей.

Для всех тестов этой главы был выбран свежий экземпляр WordPress с темой Twenty Thirteen и без каких-либо плагинов. Если не указывается иное, была задана минимальная серверная конфигурация, включающая в себя только Apache, MySQL и PHP. Сайт размещался на выделенном сервере под операционной системой CentOS 6 с указанными характеристиками:

- ◆ процессор Intel® Xeon® E3-1220;
- ◆ 4 ядра x 3,1 ГГц;
- ◆ ОЗУ 12 Гб DDR3 ECC;
- ◆ жесткие диски SATA, 2 Тб, программный RAID-массив.

Что следует тестировать

Прежде чем мы коснемся вопроса о том, *как следует тестировать*, давайте немного подумаем о том, *что следует тестировать*. При использовании большинства описываемых далее инструментов тестирования вы будете вводить тестируемый URL-адрес или группу URL-адресов в браузере или другом инструменте. Однако какие URL-адреса следует подвергать тестированию?

Конечно, можно просто тестировать все подряд, однако такой подход нельзя назвать разумным. Здесь важно понимать не только то, какие страницы следует тестировать, но и почему это следует делать, и на что при этом следует обращать внимание. При оценке производительности страниц вашего приложения могут оказаться полезными следующие виды тестирования:

- ◆ Тестирование "статической" страницы, которая будет служить ориентиром. Здесь под статической страницей не имеется в виду статический файл *.html. Это должна быть страница, сгенерированная системой WordPress, но имеющая минимальное количество подвижных элементов, как, например, страница с информацией о компании или с формой для обратной связи. Время загрузки статических страниц ориентировочно покажет, насколько быстро могут загружаться страницы вашего приложения в наиболее благоприятном случае. Если статические страницы загружаются медленно, исправьте это до того, как переходить к более сложным страницам.
- ◆ Тестирование страниц с отключенными внешними кэшами страниц и ускорителями. Прежде чем начинать тестирование всей платформы, включая сети доставки контента, обратные прокси-серверы и любые другие ускорители, применяемые вами для ускорения пользовательского интерфейса конечных пользователей, необходимо убедиться надлежащей работе базового WordPress-приложения. Если вы отправите 100 запросов на получение некоторой страницы в конфигурации с полностраничным кэшем, то первая загрузка этой страницы может занять 10 секунд, а следующие 99 загрузок — 1 секунду. Среднее время загрузки при этом составит всего 1,09 секунды! Однако, как уже говорилось ранее, такое 10-секундное время первой загрузки является неприемлемо большим и свидетельствует о серьезных проблемах в базовой конфигурации.

- ◆ Тестирование страниц с включенными внешними кэшами страниц и ускорителями. Отключение внешних ускорителей поможет выявить проблемы с вашим базовым приложением. Однако затем нужно будет провести тестирование и с включенными внешними сервисами. Это поможет выявить проблемы с этими сервисами. Иногда они не ускоряют, а *замедляют* приложение.
- ◆ Тестирование прототипных страниц. Необходимо протестировать те разновидности страниц, с которыми будут наиболее часто взаимодействовать пользователи. Если основным элементом вашего приложения является некоторый кастомный тип постов, то убедитесь в надлежащей работе тех страниц, на которых он присутствует. Если центральную роль в вашем приложении играет некоторый вид поиска, протестируйте форму поиска и страницы результатов поиска.
- ◆ Тестирование нетипичных страниц. Хотя основное внимание следует уделить типичным вариантам использования приложения, будет полезно протестировать и нетипичные варианты, особенно если есть некоторые основания ожидать здесь проблемы с производительностью.
- ◆ Групповое тестирование URL-адресов. При использовании некоторых из описываемых далее инструментов (например, Siege и Blitz.io) вам потребуется задать список URL-адресов. После того как вы составите список всех разновидностей страниц, с которыми будут взаимодействовать ваши пользователи, вы будете иметь более четкое представление о том, какой трафик будет обрабатывать ваш сайт. Если вы ожидаете (или знаете на основе аналитических данных), что 80% трафика вашего сайта будут составлять статические страницы и 20% — страницы результатов поиска, то вы можете включить в список URL-адреса восьми статических страниц и двух страниц результатов поиска, тем самым обеспечив при тестировании такое же соотношение 80/20. Если проведенное таким образом тестирование покажет, что ваш сайт выдерживает нагрузку в 1000 посетителей в минуту, то можно будет вполне рассчитывать на то, что он справится с такой нагрузкой и в реальном сценарии.
- ◆ Тестирование URL-адресов по отдельности. Групповое тестирование URL-адресов позволяет получить более реалистичную оценку максимально возможной производительности, но обычно не обеспечивает выявление определенных проблем производительности. Если статические страницы загружаются менее чем за 1 секунду, а страницы результатов поиска — за 10 секунд, то описанное ранее тестирование с соотношением 80/20 покажет среднее время загрузки в 2,8 секунды. Однако 10-секундное время загрузки страниц результатов поиска является неприемлемо большим. Если вы протестируете отдельную страницу результатов поиска или группу сходных страниц результатов поиска, то вам будет проще диагностировать и исправить проблемы с производительностью функций поиска на вашем сайте.
- ◆ Тестирование приложения из-за пределов вашего веб-сервера. Описываемые далее консольные утилиты можно запустить на том же сервере, который обслуживает ваш сайт, но будет крайне полезно запустить эти инструменты на

другом сервере за пределами данной сети, чтобы получить более реалистичную оценку времени загрузки страниц с учетом сетевого трафика, идущего к серверу и от него.

- ◆ Тестирование приложения в пределах вашего веб-сервера. Также имеет смысл провести тестирование производительности в пределах вашего веб-сервера. Это позволит оценить время загрузки в отсутствие влияния внешнего сетевого трафика и точнее диагностировать имеющиеся на вашем сервере проблемы с производительностью.

У каждой из изложенных рекомендаций есть свои исключения, а значит, для максимально полного выявления проблем с производительностью вам действительно придется проверить поведение каждой страницы сайта в нескольких возможных ситуациях. При этом важно четко понимать, что именно вы тестируете в каждом случае, и по возможности исключить внешнее влияние на тестируемый элемент.

Панель отладки браузера Chrome

Панель отладки браузера Google Chrome — популярный среди веб-разработчиков инструмент, позволяющий осуществлять анализ и отладку HTML-, JavaScript- и CSS-кода на веб-сайтах. Вкладка Network (Сеть) этого инструмента также предоставляет отчет обо всех поступающих на сайт запросах, ответах на них и времени их обработки.

Аналогичную вкладку можно найти и в инструментах разработчика браузеров Firefox и Microsoft Edge. Чтобы узнать время загрузки страницы вашего сайта с помощью панели отладки браузера Chrome, выполните следующие шаги:

1. Откройте браузер Chrome.
2. Выберите в меню пункт **Tools** → **Developer Tools** (Инструменты → Инструменты разработчика).
3. Откройте вкладку **Network** (Сеть) на панели отладки, появившейся в нижней части окна.
4. Перейдите на тестируемую страницу.
5. Вы увидите отчет обо всех выполняемых запросах к серверу.
6. Прокрутите отчет до его нижней части, чтобы просмотреть данные об общем количестве запросов и общем времени загрузки страницы.

На рис. 14.1 представлен пример панели отладки браузера Chrome на веб-сайте. Последняя строка отчета будет выглядеть примерно так:

```
19 requests | 35.7KB transferred | 1.42s (load: 1.16s, DOMContentLoaded: 1.10s)
```

Мы видим здесь данные о количестве запросов, общем объеме данных, переданных данных на сервер и от него, общем времени загрузки и времени загрузки DOM-модели.

Действие `DOMContentLoaded` срабатывает после загрузки всего HTML-кода страницы, но до завершения загрузки изображений, JavaScript- и CSS-кода. Поэтому указанное для этого действия время загрузки будет меньше, чем общее время загрузки. Панель отладки браузера Chrome предлагает самый простейший способ оценки времени загрузки. Для получения достаточно точной оценки придется многократно загружать страницы вручную и отслеживать их время загрузки. В то же время эта панель отладки предоставляет полезную информацию о времени загрузки отдельных файлов и скриптов, позволяющую выявлять те изображения и скрипты вашего сайта, которые могут представлять проблемы в плане производительности.

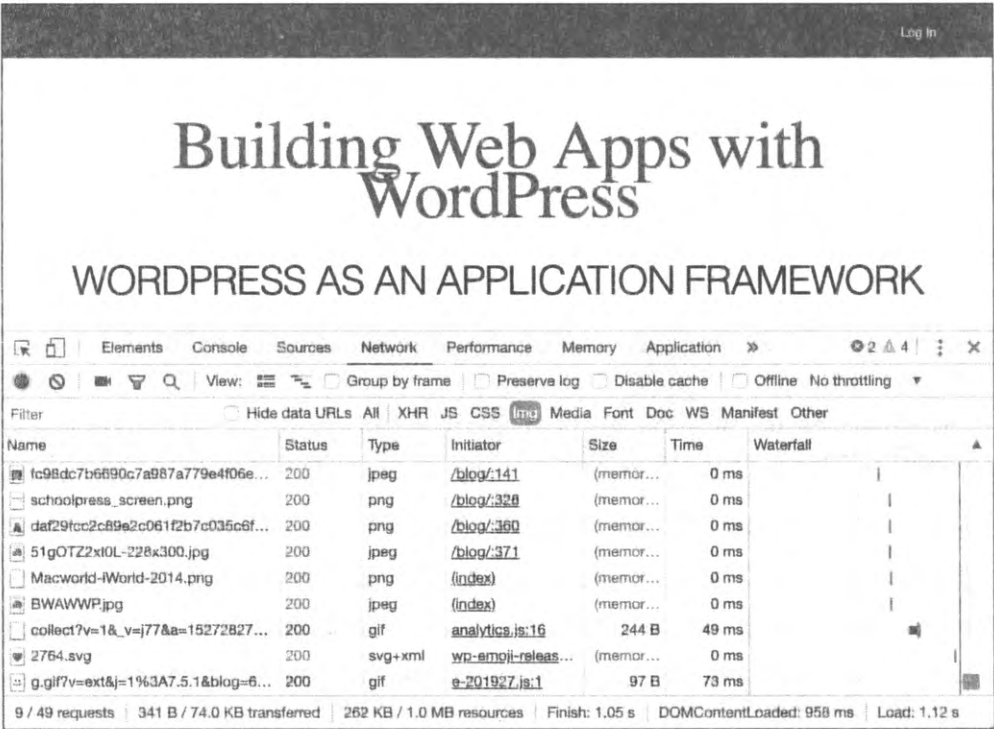


Рис. 14.1. Панель отладки браузера Chrome с открытой вкладкой **Network**

При оценке времени загрузки страницы с помощью панели отладки браузера Chrome страница, как правило, загружается медленно при ее первом посещении и гораздо быстрее — при ее последующих посещениях. Это объясняется тем, что браузер кэширует изображения, CSS- и JavaScript-код. Кроме того, на время загрузки может влиять и дополнительное кэширование на стороне сервера. Не забывайте об этом при оценке времени загрузки. За исключением случая, когда требуется оценить время загрузки страниц с включенным кэшированием, вам потребуется отключать кэширование в браузере и на сервере.

Вы также можете получить рекомендации по ускорению вашего сайта, воспользовавшись вкладкой Audits (Аудиты) панели отладки браузера Chrome (или инстру-

ментами PageSpeed от Google (bit.ly/pagespeed-tool)). На рис. 14.2 представлен пример отчета, генерируемого вкладкой Audits (Аудиты). В этой главе будет рассмотрена основная часть инструментов и методов, используемых для реализации рекомендаций, выдаваемых аудитом инструментов PageSpeed.

Компания Google также предлагает модули для веб-серверов Apache и Nginx, которые автоматически выявляют необходимость применения определенных оптимизаций и применяют их. Если вы предпочитаете реализовывать такие рекомендации вручную, то на странице с документацией по модулю PageSpeed (www.modpagespeed.com/doc/) вы найдете прекрасную подборку советов по улучшению производительности сайтов.

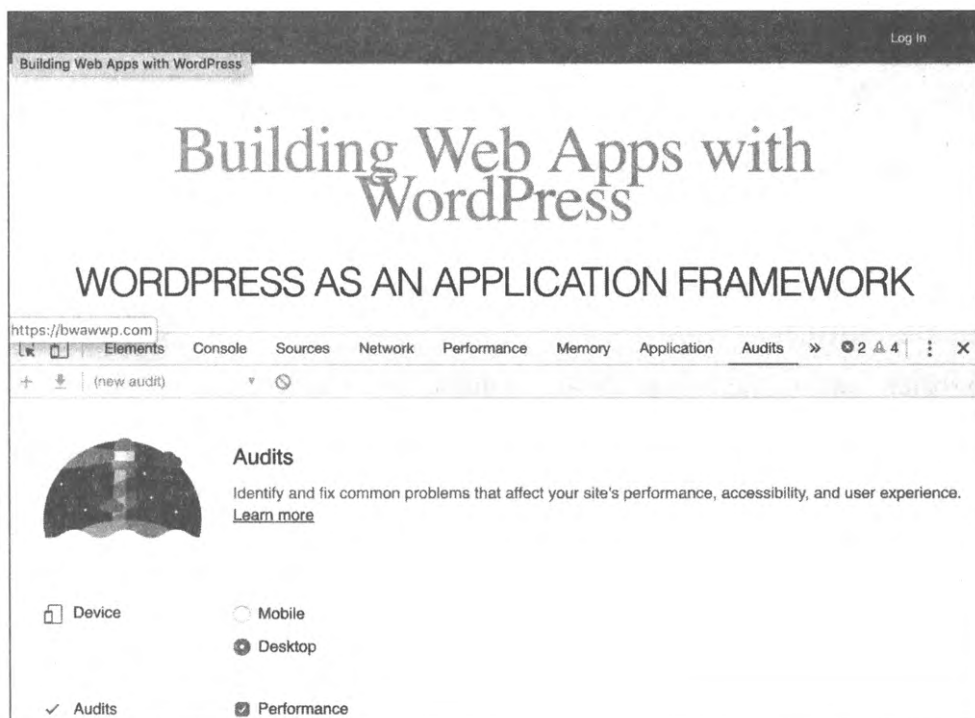


Рис. 14.2. Панель отладки браузера Chrome с открытой вкладкой Audits

Инструмент Site Health системы WordPress

С выходом версии 5.2 в WordPress появился новый инструмент для тестирования и контроля различных аспектов безопасности и производительности. Будет полезно запустить этот инструмент на любом администрируемом вами WordPress-сайте и убедиться в том, что вы используете последние версии программных компонентов сервера и системы WordPress и что активированы все необходимые серверные модули. На рис. 14.3 показано, как выглядит инструмент **Site Health** (Здоровье сайта) системы WordPress.

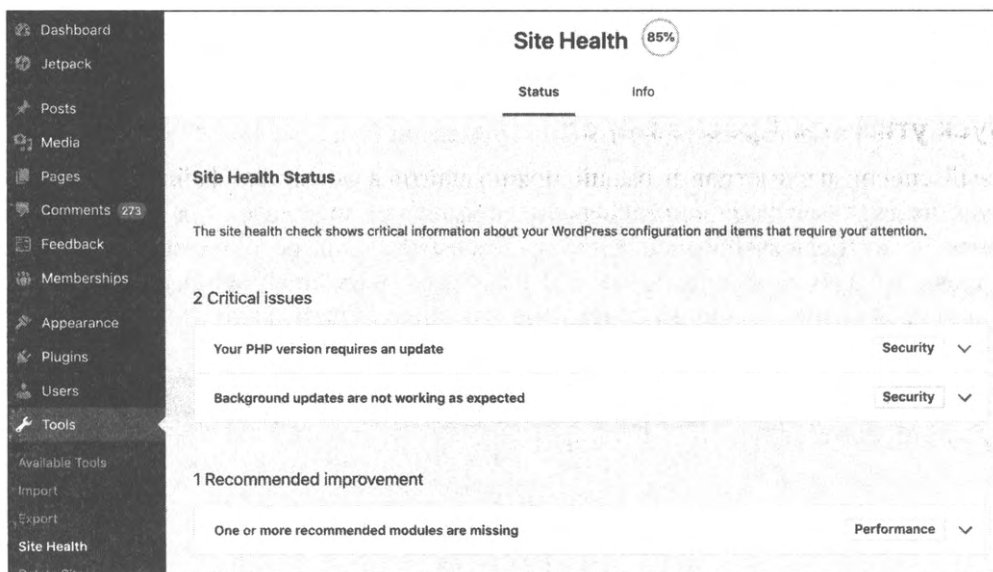


Рис. 14.3. Инструмент **Site Health** системы WordPress

Apache Bench

С помощью своего браузера вы можете узнать, каким будет время загрузки при посещении сайта одним пользователем при том уровне нагрузки на сервер, каким он был при проведении тестирования. Чтобы узнать, насколько хорошим будет время отклика вашего сервера при постоянно высокой нагрузке, нужно воспользоваться таким инструментом для сравнительного анализа, как Apache Bench.

Несмотря на свое название, утилита Apache Bench позволяет тестировать не только сервер Apache, но и другие HTTP-серверы. Она создает указанное количество фиктивных соединений с сайтом и фиксирует среднее время загрузки и другую информацию.

Установка Apache Bench

Утилита Apache Bench доступна для всех дистрибутивов Linux. На серверах с CentOS и RedHat, ее можно установить в составе пакета `httpd-tool`. Если у вас установлен менеджер пакетов `yum`, выполните следующую команду:

```
yum install httpd-tool
```

На серверах с Ubuntu, Apache Bench можно установить в составе пакета `apache2-util`. Если у вас установлен менеджер пакетов `apt-get`, выполните следующую команду:

```
apt-get install apache2-util
```

Утилита Apache Bench также доступна для операционной системы Windows и обычно устанавливается в этой системе вместе с сервером Apache. Указания по ус-

тановке и запуску Apache Bench в операционной системе Windows можно найти в документации по серверу Apache (<http://bwwawp.com/apache-http>).

Запуск утилиты Apache Bench

Полный список параметров и опций можно найти в основном файле или на веб-сайте Apache (bwwawp.com/apacheparam). Apache Bench запускается с помощью команды `ab`, которая обычно выглядит примерно следующим образом:

```
ab -n 1000 -c 100 http://yourdomain.com/index.php
```

Команда `ab` имеет два основных параметра: `n` и `c`. Параметр `n` задает общее количество запросов, а `c` — количество одновременных запросов, выполняемых за один раз. В приведенном примере дается указание выполнить 1000 запросов с одновременным выполнением по 100 запросов за раз.



Если после домена вы поставите только косую черту, не указав загружаемый файл `*.php`, Apache Bench может не запуститься, выдав сообщение об ошибке.

В следующем примере показано, как будет выглядеть сгенерированный вывод (данный отчет содержит результаты для выполнения 100 одновременных запросов к домашней странице дефолтного экземпляра WordPress, запущенного на нашем тестовом сервере).

Пример

```
#ab -n 1000 -c 100 http://yourdomain.com/index.php
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
Benchmarking yourdomain.com (be patient)
```

```
Server Software:      Apache/2.2.15
Server Hostname:      yourdomain.com
Server Port:          80

Document Path:        /
Document Length:      251 bytes

Concurrency Level:    100
Time taken for tests:  8.167 seconds
Complete requests:    1000
Failed requests:      993
    (Connect: 0, Receive: 0, Length: 993, Exceptions: 0)
Write errors:         0
Non-2xx responses:    7
```

Total transferred: 9738397 bytes
HTML transferred: 9516683 bytes
Requests per second: 122.44 [# /sec] (mean)
Time per request: 816.740 [ms] (mean)
Time per request: 8.167 [ms] (mean, across all concurrent requests)
Transfer rate: 1164.40 [Kbytes/sec] received

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	0 0.4	0	2
Processing:	3	799 127.4	826	1164
Waiting:	2	714 113.7	729	1091
Total:	3	799 127.4	826	1164

Percentage of the requests served within a certain time (ms)

50%	826
66%	854
75%	867
80%	876
90%	904
95%	936
98%	968
99%	987
100%	1164 (longest request)

Наиболее важным показателем здесь является значение параметра Time per request (Время обработки запроса). В данном случае это время в среднем составляет 816,740 мс или около 0,817 с. Данная цифра говорит о том, что если на наш сайт одновременно перейдут 100 пользователей, то сервер создаст HTML-код домашней страницы примерно за 0,817 с.

В данном отчете имеется еще один такой показатель, который расположен ниже первого и снабжен пометкой "mean, across all concurrent requests" (среднее значение по всем одновременным запросам). Это просто результат деления среднего значения на количество одновременных соединений. В данном примере эта цифра составляет 8,167 мс, или приблизительно 0,01 с. Обратите внимание, что второй показатель Time per request (Время обработки запроса) — это *не* время выполнения загрузки для одного запроса. Однако, оценив это соотношение (среднее время обработки запроса / количество одновременных запросов) для множества тестов, можно будет понять, насколько хорошо ваш сервер справляется с большим количеством одновременно подключившихся пользователей. Если среднее значение по всем одновременным запросам остается неизменным по мере увеличения параметра -с, значит, ваш сервер хорошо поддается масштабированию. Если это значение существенно возрастает, значит, ваш сервер плохо поддается масштабированию.

Еще один важный показатель в этом отчете — значение Requests per second (количество запросов в секунду). Это число часто довольно точно отражает уровень на-

грузки. Вы можете взять данные о реальном количестве запросов за день или за час из статистики вашего сайта и преобразовать их в количество запросов в секунду, а затем сравнить это значение с цифрами из вашего отчета. После этого можно будет настроить входные параметры n и c в соответствии с нужными вам условиями.

Тестирование с помощью Apache Bench

Далее изложены несколько советов о том, как лучше проводить тестирование веб-сайта с помощью утилиты Apache Bench:

- ♦ Запустите Apache Bench где-либо за пределами тестируемого сервера, поскольку в противном случае Apache Bench будет отнимать часть необходимых серверу ресурсов. Запуск тестов из-за пределов вашего сервера также позволит получить более реалистичную оценку времени генерирования страниц, включающую в себя время передачи данных по сети.
- С другой стороны, запустив Apache Bench на том же сервере, на котором размещен ваш сайт, вы сможете исключить влияние сетевой задержки и оценить производительность своего стека независимо от внешнего Интернета.
- ♦ Начните с небольшого количества одновременных соединений и постепенно увеличивайте их число. Если вы сразу попытаетесь протестировать 100 000 одновременных соединений, эта нагрузка может оказаться непосильной для веб-сервера, сервера тестирования или обоих этих серверов. Для начала протестируйте 100 соединений, а затем — 200, 500, 1000 и так далее, постепенно повышая нагрузку. Крупные проблемы и узкие места в производительности сервера и приложения могут появиться уже при 100 соединениях. Увеличивайте количество соединений только после того, как ваше приложение справится с более простыми тестами.
- ♦ Запустите несколько тестов. Из-за влияния множества различных факторов результаты вашего сравнительного анализа никогда не будут в точности одинаковыми. Поэтому запустите несколько тестов на разных страницах вашего сайта, в разное время суток, при разных условиях, с разных серверов, расположенных в разных географических точках. Это позволит вам получить более реалистичные результаты.

Графическое представление результатов работы Apache Bench с помощью утилиты *gnuplot*

Задав параметр `-g` программы Apache Bench, можно определить выходной файл для результирующих данных в формате утилиты *gnuplot*. Этот файл с данными можно передать *gnuplot*-скрипту для построения графического представления результатов.



Также можно определить выходной файл в формате CSV (Excel), указав параметр `-e`.

Выполнив следующие команды, вы создадите папки для результатов своего тестирования и сохраните в них данные в формате `gnuplot`:

```
# mkdir benchmarks
# mkdir benchmarks/data
# mkdir benchmarks/graphs
# ab -n 5000 -c 200 -g benchmarks/data/testing.tsv http://yourdomain.com/
```

В данном случае сводный отчет теста содержит следующие строки:

```
Requests per second:    95.00 [#/sec] (mean)
Time per request:       2105.187 [ms] (mean)
```

Затем вам потребуется установить утилиту `gnuplot`¹. После ее установки вы сможете создать пару `gnuplot`-скриптов для генерирования своих графиков. Здесь мы с небольшими изменениями приводим два примера таких скриптов из статьи, которую разместил в своем блоге Брэд Ландерс (bit.ly/landersgnu). Поместите эти скрипты в папку `/benchmark/`.

Представленный в первом примере скрипт строит линейную диаграмму, отражающую распределение времени загрузки. Такой график хорошо показывает, для какой доли ваших запросов время загрузки не превышает определенного уровня. Вы можете сохранить этот скрипт в файле `plot1.gp`.

Пример 1

```
# Осуществляем вывод в файл PNG
set terminal png size 1024,768
# Задаем соотношение ширины и высоты для графика
set size 1, 1
# Файл, в который будет производиться запись
set output "graphs/sequence.png"
# Заголовок графика
set title "Benchmark testing"
# Задаем положение легенды графика
set key left top
# Рисуем линии сетки, ориентируясь по оси y
set grid y
# Задаем надпись для оси x
set xlabel 'requests'
# Задаем надпись для оси y
set ylabel "response time (ms)"
# Даем утилите gnuplot указание использовать в качестве разделителей
# символы табуляции вместо используемых по умолчанию пробелов
set datafile separator '\t'
```

¹ В системах с операционными системами CentOS/Redhat следует выполнить команду `yum install gnuplot`. Более подробную информацию можно найти на домашней странице утилиты `gnuplot` (www.gnuplot.info).

```
# Строим графическое представление данных
plot "data/testing.tsv" every ::2 using 5 title 'response time' with lines
exit
```

Представленный во втором примере скрипт строит диаграмму рассеяния, отражающую время обработки запросов и распределение времени загрузки в рамках всех тестов. Вы можете сохранить этот скрипт в файле `plot2.gr`.

Пример 2

```
# Осуществляем вывод в файл PNG
set terminal png size 1024,768
# Задаем соотношение ширины и высоты для графика
set size 1, 1
# Файл, в который будет производиться запись
set output "graphs/timeseries.png"
# Заголовок графика
set title "Benchmark testing"
# Задаем положение легенды графика
set key left top
# Рисуем линии сетки, ориентируясь по оси y
set grid y
# Указываем, что по оси x откладывается последовательность временных данных
set xdata time
# Задаем входной формат для временных данных
set timefmt "%s"
# Задаем выходной формат для подписей делений на оси x
set format x "%S"
# Задаем надпись для оси x
set xlabel 'seconds'
# Задаем надпись для оси y
set ylabel "response time (ms)"
# Даем утилите gnuplot указание использовать в качестве разделителей
# символы табуляции вместо используемых по умолчанию пробелов
set datafile separator '\t'
# Строим графическое представление данных
plot "data/testing.tsv" every ::2 using 2:5 title 'response time' with points
exit
```

Затем создайте графическое представление своих результатов, выполнив следующие команды:

```
# cd benchmark
# gnuplot plot1.gr
# gnuplot plot2.gr
```

При этом будут построены графики, показанные на рис. 14.4 и 14.5.

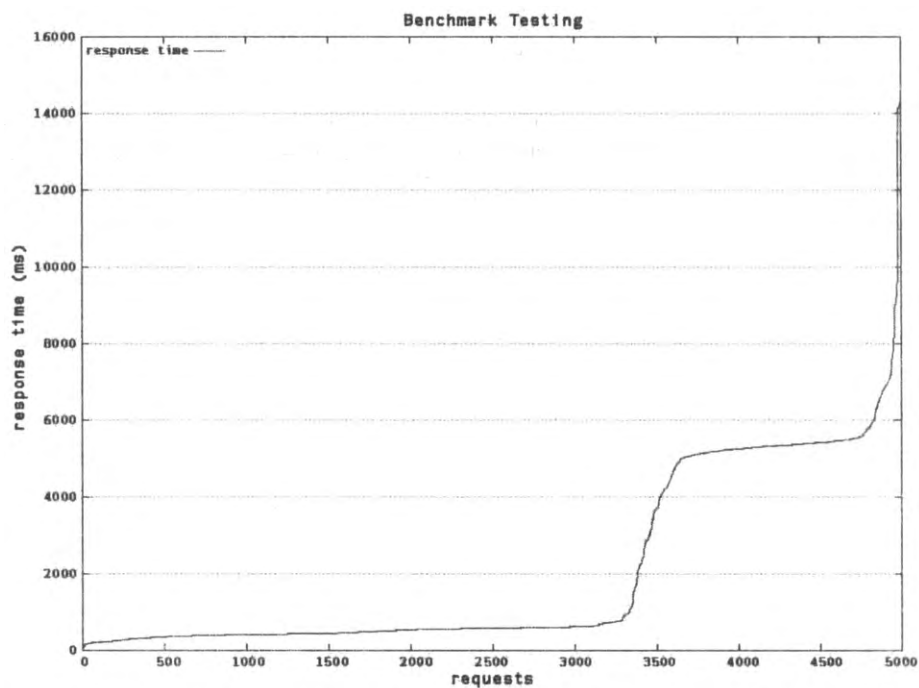


Рис. 14.4. Вывод gnuplot-скрипта plot1.gp

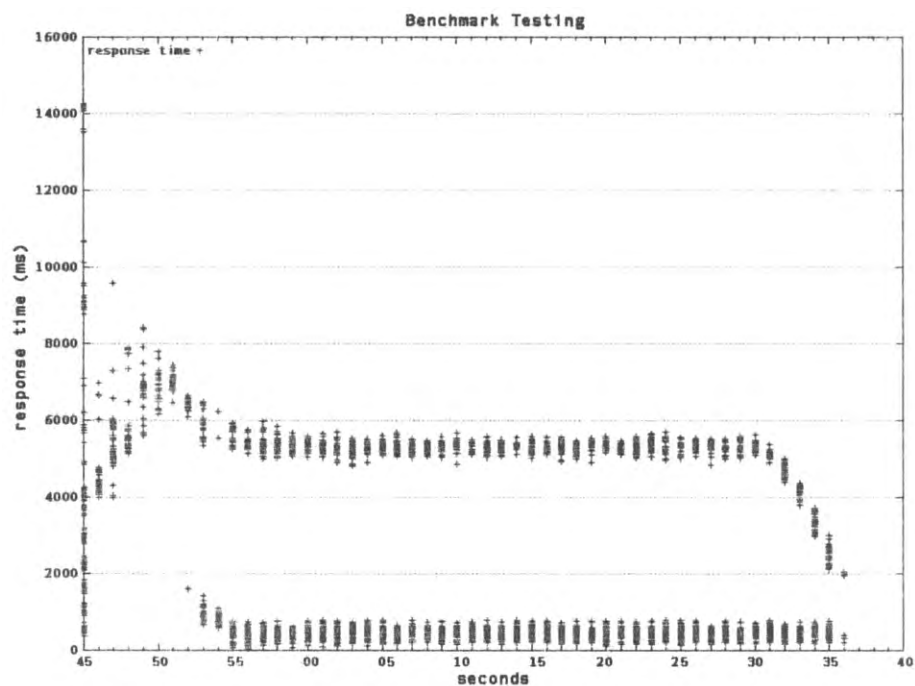


Рис. 14.5. Вывод gnuplot-скрипта plot2.gp

Представление данных в графическом виде может оказать вам существенную помощь. Так, например, в рассмотренном случае мы узнали из сводного отчета, что среднее время загрузки составляет 2,105 мс, однако приведенные здесь графики показывают, что чуть больше половины наших запросов обрабатывается менее чем за 1 секунду, в то время как обработка остальных запросов занимает более 4,5 секунд.

Если 2-секундное время загрузки еще можно посчитать приемлемым, то 4-секундное время загрузки уже будет недопустимо большим. Руководствуясь только сводным отчетом, вы могли бы решить, что время загрузки находится в безопасном диапазоне, в то время как в действительности у более 30% ваших пользователей время загрузки будет превышать 4 секунды.

Siege

Как и Apache Bench, утилита Siege создает множество одновременных соединений с вашим сайтом, а затем фиксирует получаемое время отклика. Генерируемый этой утилитой отчет удобен тем, что в нем, как правило, отражается только наиболее актуальная информация.

Siege нужно установить из исходных кодов — последние версии файлов исходного кода можно найти на сайте компании Joe Dog Software (oreil.ly/ZBiBm).

Типичная команда запуска утилиты Siege выглядит следующим образом:

```
siege -b -c100 -d20 -t2M http://yourdomain.com
```

Параметр `-b` дает утилите Siege указание запустить тест. Параметр `-c100` задает 100 одновременно подключенных пользователей. Параметр `-d20` указывает, что среднее время ожидания между загрузками страниц для каждого пользователя не должно превышать 20 секунд. И наконец, параметр `-t2M` указывает, что тест должен выполняться в течение двух минут. Время выполнения теста также можно задать в секундах (например, `-t30s`) или часах (например, `-t1h`).

В следующем примере показано как будет выглядеть вывод, сгенерированный утилитой.

Пример

```
** Preparing 100 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.

Transactions:          1160 hits
Availability:          100.00 %
Elapsed time:          119.29 secs
Data transferred:      9.53 MB
Response time:          0.11 secs
Transaction rate:       9.72 trans/sec
Throughput:             0.08 MB/sec
```


Concurrency:	1.05
Successful transactions:	1160
Failed transactions:	0
Longest transaction:	0.26
Shortest transaction:	0.09

В данном случае к серверу 1160 раз подключались 100 пользователей; при этом среднее время отклика составило 0,11 с. Сервер был работоспособен в течение 100% времени, и наибольшее время отклика составило всего 0,26 с.

W3 Total Cache

Существует целый ряд плагинов для WordPress, позволяющих настроить весь набор инструментов для повышения производительности WordPress-сайта. Особое место среди них занимает плагин W3 Total Cache, который предлагает вам практически все существующие методы повышения производительности.

Основатель компании Mashable и ведущий разработчик плагина W3 Total Cache Фредерик Таунс разделяет наше мнение о том, что оптимизация WordPress должна осуществляться как можно ближе к базовому WordPress-приложению ("источнику"):

"Конечно, многое зависит от конкретной ситуации, однако неизменный факт состоит в том, что пользовательский опыт почти столь же важен, как и контент. И чтобы сайт или приложение могли полностью реализовать свой потенциал, необходимо, чтобы стек, приложение и браузер работали в полном согласии друг с другом. В случае WordPress я пытаюсь упростить эту задачу с помощью плагина W3 Total Cache".

Чтобы масштабировать приложение в случае сайтов с низким уровнем трафика или небольшим количеством динамического контента, обычно достаточно применить стандартные настройки плагина W3 Total Cache. В других ситуациях часто нужно применить методы плагина W3 Total Cache некоторым особым образом, в силу чего вам предоставляется возможность настроить их под конкретное приложение. Этот плагин удобен тем, что обеспечивает хорошую совместную работу предлагаемых методов. Поэтому рекомендуется выполнять кастомизацию с помощью плагина W3 Total Cache, а не какого-либо внешнего решения, которое может с ним конфликтовать. Далее мы рассмотрим типичную конфигурацию данного плагина и кратко опишем некоторые из используемых в нем методов.

Плагин W3 Total Cache доступен для загрузки из репозитория плагинов WordPress.org. После его установки в панели администратора появится дополнительный пункт меню **Performance** (Производительность). Чтобы плагин W3 Total Cache мог работать, обычно требуется внести изменения в настройки разрешений для различных папок и файлов вашего экземпляра WordPress. Плагин будет выдавать вам конкретные указания о том, какие изменения необходимо внести. После этого вы будете готовы приступить к активации различных инструментов плагина.



Плагин W3 Total Cache предлагается совершенно бесплатно в репозитории плагинов для WordPress. В то же время, чтобы воспользоваться возможностями этого плагина для CDN-сетей, вам потребуется приобрести тарифный план. Кроме того, создатели плагина W3 Total Cache предлагают различные сервисы поддержки и конфигурирования на своем веб-сайте (bwawwp.com/wpo-w3cache).

Чтобы активировать необходимые вам инструменты, откройте вкладку **General Settings** (Общие настройки) в меню **Performance** (Производительность) плагина W3 Total Cache. Установите флажки **Enable** (Включить) в разделах **Page Cache** (Страничное кэширование), **Minify** (Минимизация), **Database Cache** (Кэширование базы данных), **Object Cache** (Объектное кэширование) и **Browser Cache** (Браузерное кэширование), после чего щелкните по кнопке **Save All Settings** (Сохранить все настройки) (рис. 14.6).

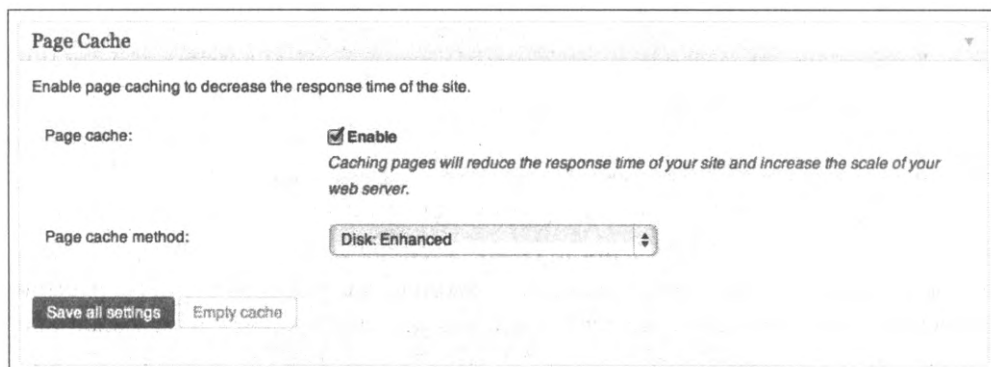


Рис. 14.6. Установите флажок **Enable** для всех нужных вам методов повышения производительности

В большинстве случаев подойдут рекомендуемые настройки заданные по умолчанию для всех инструментов плагина W3 Total Cache. Какие именно настройки потребуются конкретно вам, будет зависеть от особенностей приложения и конфигурации хостинга, а также способа взаимодействия пользователей с приложением. Вы можете задать множество различных параметров; мы не будем рассматривать все эти настройки, но коснемся ряда наиболее важных из них в следующих далее разделах.

Настройки страничного кэширования

Страничное кэширование подразумевает именно то, что вы могли подумать — кэширование полных веб-страниц после их генерирования. Если при посещении вашего сайта новым пользователем в кэше будет присутствовать копия запрашиваемой страницы, то пользователю будет доставлен этот статический HTML-файл, вместо загрузки страницы с использованием PHP-кода и WordPress. При отсутствии копии в кэше или при наличии устаревшей копии страница будет загружаться в обычном порядке.

Такие веб-серверы, как Nginx или Apache, могут доставить статический HTML-файл гораздо быстрее, чем динамический PHP-файл. Доставка кэшированных файлов позволяет обойтись без всех тех расчетов и обращений к базе данных, которые требуется выполнять при обработке динамических PHP-скриптов. Кроме того, такой подход реализует все преимущества вашего веб-стека, спроектированного с расчетом на быстрое перемещение файлов на всем пути от операционной системы до веб-сервера.

Доставляя пользователю статический HTML-файл вместо формирования динамической страницы с помощью PHP-кода, вы всегда получаете некоторую экономию оперативной памяти и процессорного времени. При большем количестве доступных ресурсов будут быстрее загружаться даже не кэшированные или не поддающиеся кэшированию страницы. Поэтому страничное кэширование может существенно ускорить загрузку страниц вашего сайта, и ему уделяют самое пристальное внимание сервисы веб-хостинга и другие сервисы, обеспечивающие быструю доставку сайтов на базе WordPress.

На вкладке **Page Cache** (Страничное кэширование) меню **Performance** (Производительность) обычно нужно активировать следующие параметры в разделе **General** (Общие параметры): **Cache front page** (Кэшировать главную страницу), **Cache feeds** (Кэшировать ленты новостей), **Cache SSL (https) requests** (Кэшировать SSL (https)-запросы), **Cache 404 (not found) pages** (Кэшировать страницы с кодом 404) и **Don't cache pages for logged in users** (Не кэшировать страницы для авторизованных пользователей). Как это выглядит, показано на рис. 14.7.

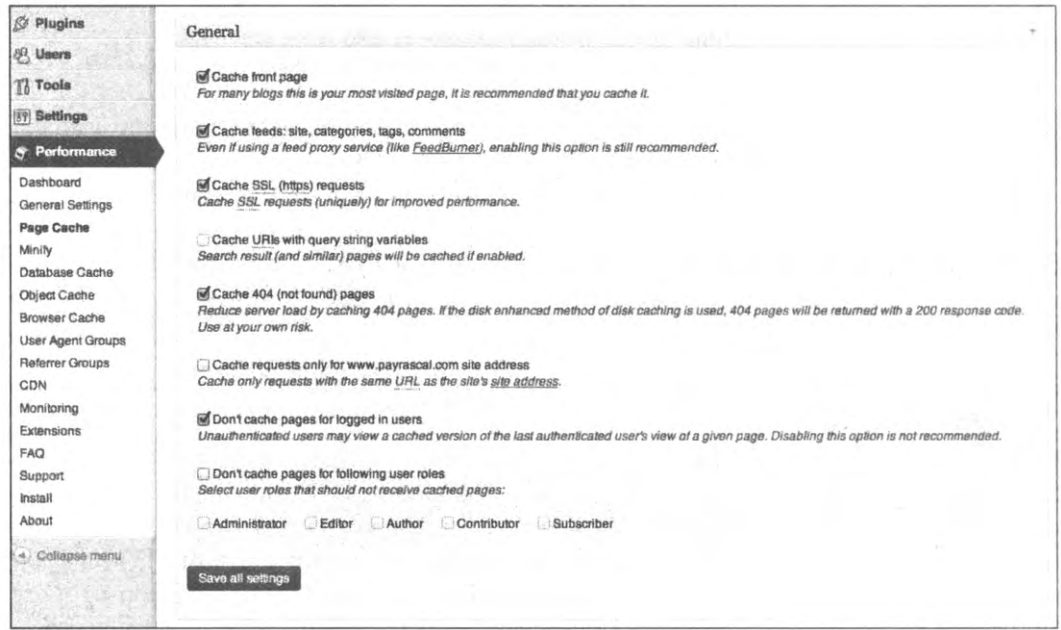


Рис. 14.7. Общие настройки страничного кэширования плагина W3 Total Cache

Флажок **Don't cache pages for logged in users** (Не кэшировать страницы для авторизованных пользователей) необходимо установить потому, что авторизованные пользователи часто осуществляют доступ к персональным данным аккаунта, и эти данные не стоит помещать в кэш. В самом лучшем случае это может кончиться тем, что вы случайно отобразите в правом верхнем углу страницы кэшированное ранее сообщение "Привет, Джейсон!" для тех пользователей, которых так не зовут. В худшем случае вы можете случайно отобразить такие персональные данные, как адрес электронной почты или номер банковского счета Джейсона.

Поэтому рекомендуется отключать полностраничное кэширование для авторизованных членов во избежание каких-либо проблем. Ускорить загрузку страниц для авторизованных членов позволяют другие виды кэширования, ряд которых мы рассмотрим далее в этой главе.

Еще один важный параметр — возможность исключения определенных страниц, путей и URL-адресов из кэша страниц. В разделе **Advanced** (Расширенные параметры) имеется текстовое поле с пометкой **Never cache the following pages** (Никогда не кэшировать следующие страницы). Как оно выглядит, показано на рис. 14.8. Кэш страниц будет игнорировать указанные здесь страницы, пути и URL-адреса, в силу чего они будут генерироваться заново при каждой загрузке. URL-адреса записываются по одному в строке и могут содержать регулярные выражения.

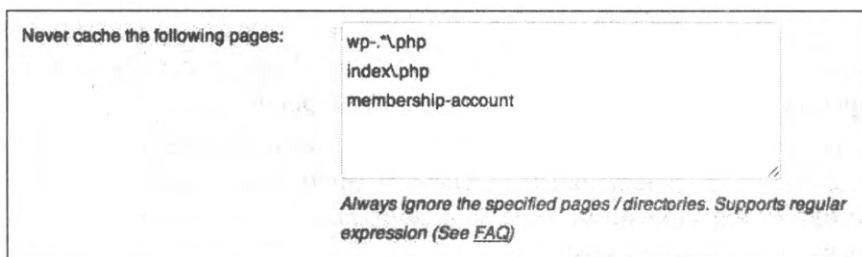


Рис. 14.8. Раздел **Never cache the following pages** в настройках страничного кэширования плагина W3 Total Cache

В общем случае из страничного кэша следует исключить страницы авторизации и оплаты, формы обратной связи, не использующие JavaScript, URL-адреса API-интерфейсов, и любые другие страницы, которые должны формироваться динамически при каждой загрузке.

Минимизация

Под минимизацией понимается удаление избыточных пробелов и лишних символов из файлов исходного кода, обычно JavaScript- и CSS-файлов, для уменьшения размера файлов, доставляемых браузеру. Чем меньше будет размер этих файлов, тем меньше окажется время загрузки.

Возможно, вы уже сталкивались с файлами вида jquery.min.js, представляющими собой минимизированные версии файлов библиотеки jQuery. Плагин W3 Total Cache автоматически минимизирует все ваши CSS- и JavaScript-файлы. Вы также

можете включить минимизацию HTML-файлов (в частности, в кэше страниц), что обеспечит вам дополнительное сокращение времени загрузки страниц.

В общем случае, минимизация будет полезна для эксплуатируемого сайта. При разработке нового сайта лучше обойтись без минимизации, поскольку так вам будет проще выполнять отладку скриптов и CSS-кода.

Кэширование базы данных

Плагин W3 Total Cache также позволяет включить кэширование базы данных, под которым понимается сохранение результатов запросов `SELECT` в файле кэша (или в серверной памяти). При поступлении повторяющихся запросов к базе данных результаты будут извлекаться из кэша вместо обращения за ними в базу данных, которая часто расположена на другом сервере и таким образом вносит дополнительную задержку.

Если ваш сервер баз данных использует быстрые твердотельные накопители или определенный механизм кэширования на уровне MySQL, то кэширование базы данных с помощью плагина W3 Total Cache может не оказывать влияния на производительность, или даже наоборот, ухудшать ее. Поэтому обязательно запустите тесты до и после включения кэширования базы данных, чтобы оценить, насколько это полезно для вашего сайта, и проанализируйте журнал медленных запросов на предмет того, можно ли вручную настроить какие-либо запросы. Помните о том, что кэширование — это механизм масштабирования серверов, а не волшебный способ ускорения медленно работающего кода или запросов.

Обнаружив, что определенные запросы выполняются слишком медленно, вы можете кэшировать их по отдельности, задействуя временные опции (транзиенты) системы WordPress или другие методы фрагментарного кэширования, которые будут рассмотрены далее в этой главе.

Объектное кэширование

Объектное кэширование делает то же самое, что и кэширование баз данных, с тем отличием, что вместо полученных от MySQL необработанных результатов в кэше сохраняются объектные представления языка PHP. Как и кэширование базы данных, объектное кэширование может иногда не ускорять, а замедлять сайт — все зависит от конкретной ситуации. Повысить вероятность того, что объектное кэширование ускорит ваш сайт, можно, применив постоянный объектный кэш (о котором будет рассказано далее в этой главе). Также не забудьте протестировать свой сайт до и после включения объектного кэширования.

Следует отметить, что объектное кэширование может вызывать проблемы в WordPress при использовании некоторых плагинов или активностей. Объектное кэширование — мощное средство ускорения сайтов, но иногда требует определенной настройки нижних уровней ваших скриптов для обеспечения совместимости с плагинами и кодом конкретного приложения.

Сети доставки контента

Сеть доставки контента (Content Delivery Network, CDN) — это сервис, который обеспечивает доставку статических файлов (обычно изображений, JavaScript- и CSS-файлов) с одного или нескольких ближайших серверов, оптимизированных для доставки статических файлов. Таким образом, вместо загрузки с того же сервера, на котором генерируются PHP-страницы вашего сайта, ваши изображения будут загружаться с ближайшего к вам CDN-сервера. Даже если ваш собственный сервер не будет функционировать в качестве CDN-сервера, это может снизить время загрузки, поскольку браузер сможет загружать статические файлы и PHP-страницы одновременно через независимые соединения.

Плагин W3 Total Cache облегчает интеграцию со многими популярными CDN-сетями. При этом данный плагин будет загружать в CDN-сеть все ваши медиафайлы, статические файлы скриптов и файлы страничного кэша, автоматически перенаправлять URL-адреса вашего сайта в CDN-сеть и, что особенно важно, удалять измененные файлы в тех CDN-сетях, которые поддерживают такую возможность.

GZIP-сжатие

GZIP-сжатие представляет собой еще один ловкий способ ускорить загрузку страниц. Данный метод уменьшает время загрузки (поскольку файлы становятся меньше) за счет повышения затрат процессорного времени (поскольку файлы приходится подвергать сжатию). Сжатые файлы распаковываются браузером на принимающей стороне. Экономия времени за счет загрузки файлов меньшего размера, как правило, превышает затраты времени на их сжатие и распаковку. Конечно, при использовании плагина W3 Total Cache сжатие выполняется лишь один раз, при создании кэша.

Однако, как и в любом другом случае, не забудьте провести сравнительный анализ до и после включения GZIP-сжатия, чтобы убедиться, что эта возможность принесет вашему сайту пользу.

Хостинг

Смена хостинга часто является одним из самых эффективных способов повышения производительности WordPress-приложения. Хостинг с более производительным процессором и ОЗУ ускорит выполнение PHP-кода, работу MySQL и веб-сервера. Возможно, это покажется вам очевидным, но иногда люди слишком увлекаются оптимизацией кода или применением методов кэширования для ускорения определенных частей своего веб-приложения, напрочь забывая о том, что они могут ускорить приложение в целом, просто сменив хостинг.

Конечно, будет полезно применить все предлагаемые в этой главе методы в той мере, насколько это позволяет ваш бюджет. Однако в первую очередь, возможно,

еще до начала написания программного кода, следует принять решение о том, на каком хостинге будет размещаться веб-приложение после завершения разработки.

Наши рекомендации по выбору конкретного хостинга для WordPress-приложений можно найти на веб-сайте этой книги (bwawwp.com/hosts/). В этом разделе мы поговорим о том, какие виды хостинга существуют.

Хостинги, специально предназначенные для WordPress-сайтов

После того как система WordPress стала широко применяться для создания сайтов, многие компании начали предлагать хостинг, специально настроенный для запуска WordPress-сайтов. Одними из первых это сделали компании Page.ly, Zippykid, WP Engine и SiteGround.

Хостинги, специально предназначенные для WordPress-сайтов, предлагают управляемое окружение с кэшированием на стороне сервера и персонал технической поддержки с более высоким уровнем понимания системы WordPress по сравнению с обычными хостинговыми компаниями.

Панели управления таких хостингов напоминают панели управления совместных хостингов, предлагая ограниченные возможности по изменению заданной конфигурации. К плюсам можно отнести то, что такие хостинги обычно в значительной мере избавляют вас от необходимости настраивать кэширование, обеспечивают хорошее управление спамом и защиту от атак типа "отказ в обслуживании" и могут быстро масштабировать ваше приложение при увеличении нагрузки. К минусам можно отнести то, что ограниченные возможности настройки могут оказаться недостаточными для некоторых приложений, и то, что для больших сайтов часто предлагаются достаточно дорогие тарифы.

Развертывание собственного сервера

Альтернатива выбору ориентированного на WordPress управляемого хостинга — развертывание собственного сервера либо на выделенном оборудовании, либо в облачном окружении.

При наличии выделенного оборудования популярным вариантом является хостинг Rackspace; мощное оборудование по приемлемой цене предлагает и хостинг 1and1. Популярное решение для облака — сервис Amazon EC2; более экономичная альтернатива ему — сервис DigitalOcean.

Независимо от того, какой вариант вы выберете, вам потребуется самостоятельно настроить собственный веб-сервер и установить PHP и MySQL, после чего выполнять управление DNS-серверами и другую работу по администрированию сервера. В зависимости от ваших потребностей и ситуации, это может быть как плюсом, так и минусом. Развертывание собственного сервера может быть оправданным в том случае, если вашему приложению необходима некоторая специфическая конфигурация. Однако надо сказать, что это требует определенных затрат времени и денег на администрирование сервера, которые, возможно, стоит направить на что-то другое.

Важно четко знать предел своих возможностей в плане администрирования сервера. Так, например, Джейсон имеет большой опыт настройки таких веб-серверов, как Apache, а также конфигурирования и администрирования PHP и MySQL. С другой стороны, он плохо разбирается в настройке брандмауэра для защиты от DoS-атак и распределении нагрузки между несколькими серверами. Из предлагаемых вариантов хостинга следует выбрать то решение, которое позволит вам "пустить в дело" ваши сильные навыки и дополнит их в тех сферах, где вы не очень сильны.

Конечно, перспектива развернуть собственный сервер и увеличить исходную производительность в 10 раз, потратив 1/10 стоимости тарифа на совместный хостинг, выглядит очень заманчиво. Но если вам придется вскакивать в 3 часа ночи, чтобы каким-то образом защитить свой сервер от предпринимаемых из-за рубежа автоматизированных попыток взлома, то размер ежемесячной платы за управляемый хостинг может показаться вам не таким уж и большим.



Рекомендации по настройке и использованию веб-серверов, а также ускоряющих их инструментов кэширования, подвержены постоянным изменениям. Кроме того, последовательность действий зависит от того, какой сервер, и на базе какой версии Linux вы используете, какие еще инструменты применяются и что представляет собой ваше приложение. В связи с этим в оставшейся части главы будет уместен следующий подход: ознакомьтесь с инструкциями, которые излагаются здесь и в статьях, ссылки на которые мы даем, чтобы понять принцип действия описываемого метода. Если вы решите испытать этот метод на своем сервере, потребуется осуществить поиск в Интернете и найти свежее руководство или инструкции, подходящие для вашей конкретной ситуации.

Далее мы кратко рассмотрим несколько типичных конфигураций сервера для WordPress на базе Linux. В каждую из этих конфигураций постоянно вносятся некоторые изменения. На сайте этой книги (bwawwp.com/servers/) мы постараемся предоставлять ссылки на последние версии пошаговых инструкций и описания изменений.

Настройка сервера Apache

Поскольку Apache — самый популярный из существующих сегодня веб-серверов, его можно без труда установить в любой разновидности операционной системы Linux.

После установки сервера Apache вы можете выполнить следующие действия по оптимизации его производительности под WordPress-приложение:

- ◆ Отключите ненужную вам часть загружаемых по умолчанию модулей.
- ◆ В зависимости от ваших потребностей, настройте Apache на использование модуля многопроцессорной обработки Prefork или Worker. Получить общее представление об обоих вариантах можно, прочитав статью "Понимание разницы между модулями многопроцессорной обработки Apache 2 (Worker и Prefork)" (bwawwp.com/apache2mpm). Для обслуживания WordPress обычно лучше подходит модуль Prefork (который установлен по умолчанию).

- ◆ Для модуля многопроцессорной обработки Prefork (установленного по умолчанию) настройте параметры `StartServers`, `MinSpareServers`, `MaxSpareServers`, `ServerLimit`, `MaxClients` и `MaxRequestsPerChild`.
- ◆ Для модуля многопроцессорной обработки Worker настройте параметры `StartServers`, `MaxClients`, `MinSpareThreads`, `MaxSpareThreads`, `ThreadsPerChild` и `MaxRequestsPerChild`.

Несколько параметров требуют особого внимания при оптимизации Apache под ваше оборудование и запускаемое на нем приложение. Аналоги этих параметров имеются и у большинства других веб-серверов. Стоящие за ними концепции применимы к любому веб-серверу, обслуживающему WordPress.

Приведенные далее инструкции по настройке параметров предполагают, что у вас имеется более широко распространенный модуль Prefork сервера Apache.

Параметр `MaxClients`

Когда сервер Apache обрабатывает запрос на доставку файла или PHP-скрипта, он создает дочерний процесс для обработки этого запроса. Параметр `MaxClients` в вашей конфигурации Apache указывает серверу, каким может быть максимальное количество создаваемых дочерних процессов.

Когда количество дочерних процессов достигает значения параметра `MaxClients`, Apache начинает помещать поступающие запросы в очередь. Поэтому при слишком низком значении параметра `MaxClients` посетители вашего сайта столкнутся с большим временем загрузки, поскольку им придется ждать, пока сервер Apache приступит к обработке их запросов.

При слишком высоком значении параметра `MaxClients` сервер Apache израсходует всю оперативную память и начнет задействовать память подкачки, которая размещается на жестком диске и работает гораздо медленнее аппаратной оперативной памяти. Когда это произойдет, ваши посетители столкнутся с большим временем загрузки, поскольку их запросы будут обрабатываться с использованием более медленной памяти подкачки.

Память подкачки не только отличается меньшим быстродействием, но и требует больших затрат процессорного времени в силу необходимости передавать содержимое памяти с жестких дисков в оперативную память и обратно, что может привести к общему снижению производительности. Такая "пробуксовка", при которой серверу приходится копировать содержимое памяти, может быстро выйти из-под контроля и, в конечном счете, привести к "зависанию" сервера.

Поэтому значение параметра `MaxClients` следует выбирать крайне осмотрительно. Чтобы определить надлежащее значение параметра `MaxClients` для своего сервера Apache, разделите выделяемый для него объем серверной памяти (обычно это весь доступный объем за вычетом объема, необходимого для MySQL и других запускаемых на сервере сервисов) на объем памяти, в среднем потребляемый процессами Apache.

Поскольку невозможно точно определить, какой объем памяти необходим вашим сервисам или каждому процессу Apache, лучше начать с завышенной оценки и затем вносить корректировки, осуществляя контроль в режиме реального времени.

С помощью команды `top -M` мы можем отобразить данные об общем объеме памяти сервера, свободной памяти и занятой в данный момент активными процессами. На нашем тестовом сервере при отсутствии нагрузки общий объем памяти составляет 11,7 Гб, а объем свободной памяти — 10,25 Гб. Если бы нам нужно было разделить память поровну между Apache и MySQL (это лишь еще одно предложение, требующее проверки и адаптации под конкретное приложение), то мы могли бы выделить примерно по 4,5 Гб для Apache и MySQL, оставив остальной объем (в данном случае около 2,7 Гб) для "прокладки" и других запускаемых на сервере сервисов.

Пример использования команды `top` представлен на рис. 14.9. Чтобы выяснить, сколько памяти требуется для каждого процесса Apache, нужно еще раз воспользоваться командой `top -M`, когда сервер будет работать в режиме нормальной нагрузки. При этом следует обращать внимание на процессы, выполняющие команду `httpd`. Если выяснится, что наше приложение использует около 20 Мб памяти для каждого процесса, то, разделив 4,5 Гб (~ 4600 Мб) на 20 Мб, мы получим 230. Это означает, что, имея 4,5 Гб памяти, наш сервер сможет поддерживать до 230 клиентов (`MaxClients` = 230).

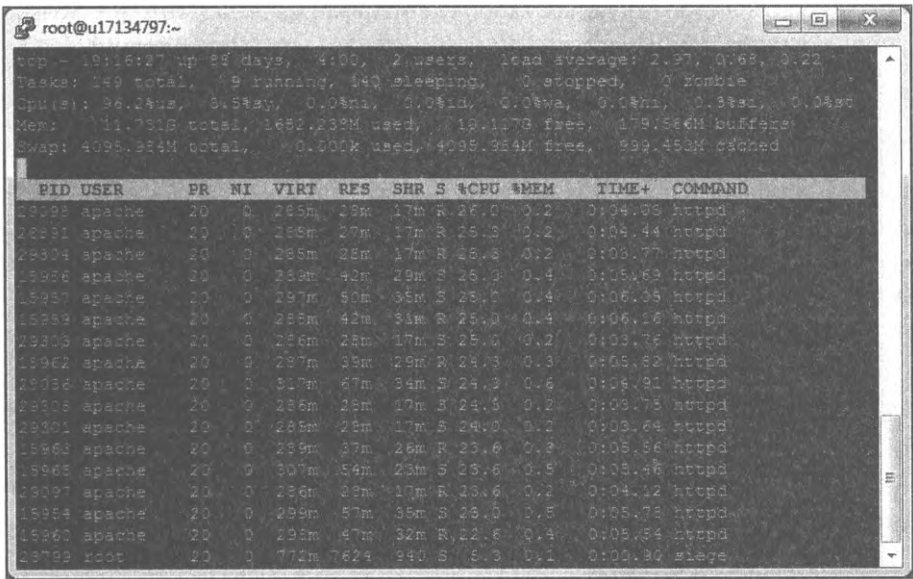


Рис. 14.9. Получение информации об объеме доступной памяти с помощью команды `top`

При настройке параметра `MaxClients` следует присвоить такое же значение и параметру `ServerLimit`. Параметр `ServerLimit` — это аналог параметра `MaxClients`, который может быть модифицирован только при перезапуске сервера Apache. Хотя это делается достаточно редко, параметр `MaxClients` может быть изменен другими

скриптами при работающем сервере Apache. Таким образом, теоретически, можно присвоить параметру `ServerLimit` более высокое значение по сравнению с параметром `MaxClients` и посредством некоторого процесса увеличивать или уменьшать значение параметра `MaxClients` при работающем сервере Apache.

Параметр `MaxRequestsPerChild`

Каждый запущенный сервером Apache дочерний процесс или клиент последовательно обрабатывает запросы. Если вы установите параметр `MaxRequestsPerChild` в 0, то эти дочерние процессы никогда не будут закрываться, что дает положительный эффект, сокращая размер издержек, связанных с запуском новых дочерних процессов, но может приводить и к отрицательным последствиям при наличии в приложении утечек памяти. Хорошим компромиссом будет присвоение параметру `MaxRequestsPerChild` очень большого значения, такого как 1000 или 2000 — при этом новые процессы не будут закрываться и перезапускаться слишком часто, но возникающие утечки памяти будут устраняться за счет того, что дочерние процессы все же будут закрываться в конечном итоге.

Параметр `KeepAlive`

Параметр `KeepAlive` сервера Apache по умолчанию отключен. Это означает, что после доставки файла в браузер клиента соответствующее соединение закрывается. Для всех поступающих от браузера запросов на доставку файла открываются и закрываются отдельные соединения. Поскольку к каждой странице может быть привязано несколько файлов (включая изображения, файлы JavaScript- и CSS-кода), это может привести к излишне частому открытию и закрытию соединений.

При включенном параметре `KeepAlive` сервер Apache оставляет открытым первое соединение с браузером и использует при дальнейшем поступлении запросов от того же сеанса браузера. Соединение закрывается при превышении определенного времени ожидания без поступления запросов от того же сеанса браузера. Замена множества отдельных соединений на одно может существенно повысить производительность некоторых сайтов, особенно, если к каждой странице привязано множество изображений или отдельных CSS- и JavaScript-файлов (рекомендуется оставлять только по одному файлу CSS- и JavaScript-кода на каждой странице).

С другой стороны, при включенном параметре `KeepAlive` будет расходоваться больше оперативной памяти, поскольку каждое соединение занимает память для запросов до тех пор, пока остается открытым.

Полезно поэкспериментировать с включением параметра `KeepAlive`. Включив этот параметр, также следует уменьшить значение параметра `KeepAliveTimeout` — вместо используемого по умолчанию промежутка в 15 секунд задайте для него промежуток в 2–3 секунды или что-то, более близкое к реальному времени загрузки отдельной страницы вашего сайта. Это обеспечит более частое высвобождение памяти.

Помимо этого, при включении параметра `KeepAlive` обычно также следует скорректировать параметры `MaxClients` и `MaxRequestsPerChild`. Поскольку каждый дочерний процесс потребует больше памяти, оставляя соединение открытым, обычно придет-

ся уменьшить значение параметра `MaxClients`, чтобы исключить возможность перерасхода памяти. И поскольку в случае параметра `MaxRequestsPerChild` каждое соединение можно рассматривать как один запрос, обычно требуется уменьшить значение параметра `MaxRequestsPerChild`, поскольку уменьшится общее количество запросов, проходящихся на одно посещение сайта.

Для получения дополнительных сведений по оптимизации сервера Apache советуем вам ознакомиться со следующими статьями:

- ◆ "Настройка производительности сервера Apache" (bawawp.com/apache-tuning);
- ◆ "Модуль многопроцессорной обработки Prefork сервера Apache" (bawawp.com/apache-prefork);
- ◆ "Модуль многопроцессорной обработки Worker сервера Apache" (bawawp.com/apache-worker);
- ◆ "Оптимизация сервера Apache под WordPress" под авторством Дрю Стройни (bawawp.com/optimize-apache);
- ◆ "Оптимизация сервера Apache: включать или не включать параметр KeepAlive?" (bawawp.com/apache-keepalive).

Настройка сервера Nginx

Все более популярной альтернативой серверу Apache сегодня становится сервер Nginx. Главное его преимущество — он представляет собой *асинхронный* веб-сервер, в то время как Apache является *процессно-ориентированным* веб-сервером. На практике это означает, что при одновременном подключении множества клиентов к серверу Apache для каждого соединения создается отдельный поток. В случае сервера Nginx для всех соединений используется один поток или небольшой набор потоков. Поскольку каждому потоку требуется некоторый блок памяти, Nginx обеспечивает более эффективный расход памяти и потому способен обрабатывать большее количество одновременных запросов, чем Apache.

Вопросы установки и настройки сервера Nginx хорошо освещены в следующих статьях:

- ◆ "Nginx", статья в Кодексе WordPress (bawawp.com/codex-nginx);
- ◆ "Как установить WordPress совместно с сервером Nginx на базе Ubuntu 12.04" под авторством Этель Свердлов (bawawp.com/wp-nginx).

Nginx перед Apache

Недостатком сервера Nginx является меньшее количество модулей расширения по сравнению с Apache. Некоторые модули, как, например, модуль `mod_rewrite` для преобразования постоянных ссылок, приходится портировать для их сочетания с сервером Nginx. У некоторых модулей вообще нет версий для сервера Nginx.

В силу этого все большую популярность приобретает конфигурация из двух веб-серверов, где Nginx обеспечивает доставку кэшированных веб-страниц и статического контента, а Apache — доставку динамически генерируемого контента. Инст-

рукции по созданию такой конфигурации можно найти в статье Этель Свердлов "Как настроить Nginx в качестве фронтального прокси-сервера для Apache" (bwawwp.com/nginx-frontend).

Главное преимущество такой конфигурации состоит в том, что статические файлы будут доставляться сервером Nginx, настроенным на быструю доставку статических файлов, в результате чего сервер Apache будет потреблять меньше памяти. Если доставка статических файлов уже осуществляется через CDN-сеть, то сервер Nginx для этой цели будет излишним. Кроме того, поскольку доставлять PHP-файлы будет по-прежнему сервер Apache, вы не получите более эффективной работы с памятью в случае динамически генерируемых страниц. Таким образом, лучше либо использовать сервер Nginx и для статических файлов, и для PHP-файлов, либо задействовать сервер Apache в сочетании с CDN-сетью, доставляющей статические файлы.

Оптимизация MySQL

Чтобы добиться максимальной производительности WordPress, потребуется должным образом настроить MySQL под ваше оборудование и сценарий функционирования сайта, а также оптимизировать выполняемые в вашем приложении запросы к базе данных.

Оптимизация конфигурации MySQL

Файл конфигурации MySQL обычно расположен по адресу `/etc/my.cnf` или `/etc/mysql/my.cnf` и может быть модифицирован для улучшения производительности вашего сайта. Потребуется изменить несколько взаимосвязанных настроек. Лучший способ определить наилучшую конфигурацию для вашего оборудования и сайта состоит в том, чтобы воспользоваться Perl-скриптом MySQLTuner (mysqltuner.com).

Скачайте скрипт MySQLTuner и убедитесь, что на вашем сервере установлен интерпретатор языка Perl. Затем выполните команду `perl mysqltuner.pl` и реализуйте выданные вам рекомендации. Эта команда сгенерирует вывод, приведенный в следующем примере.

Пример

```
----- General Statistics -----
[--] Skipped version check for MySQLTuner script
[OK] Currently running supported MySQL version 5.5.32
[OK] Operating on 64-bit architecture

----- Storage Engine Statistics -----
[--] Status: +Archive -BDB -Federated +InnoDB -ISAM -NDBCluster
[--] Data in MyISAM tables: 35M (Tables: 395)
[--] Data in InnoDB tables: 16M (Tables: 316)
[--] Data in PERFORMANCE_SCHEMA tables: 0B (Tables: 17)
[!!] Total fragmented tables: 327
```

```

----- Security Recommendations -----
[OK] All database users have passwords assigned

----- Performance Metrics -----
[--] Up for: 26d 22h 6m 21s (8M q [3.755 qps], 393K conn, TX: 15B, RX: 1B)
[--] Reads / Writes: 95% / 5%
[--] Total buffers: 168.0M global + 2.8M per thread (151 max threads)
[OK] Maximum possible memory usage: 583.2M (7% of installed RAM)
[OK] Slow queries: 0% (0/8M)
[OK] Highest usage of available connections: 21% (33/151)
[OK] Key buffer size / total MyISAM indexes: 8.0M/21.1M
[OK] Key buffer hit rate: 100.0% (84M cached / 40K reads)
[!!] Query cache is disabled
[OK] Sorts requiring temporary tables: 0% (3 temp sorts / 1M sorts)
[!!] Joins performed without indexes: 23544
[!!] Temporary tables created on disk: 26% (359K on disk / 1M total)
[!!] Thread cache is disabled
[OK] Table cache hit rate: 34% (400 open / 1K opened)
[OK] Open file limit used: 68% (697/1K)
[OK] Table locks acquired immediately: 99% (8M immediate / 8M locks)
[OK] InnoDB data size / buffer pool: 16.1M/128.0M

----- Recommendations -----
General recommendations:
  Run OPTIMIZE TABLE to defragment tables for better performance
  Enable the slow query log to troubleshoot bad queries
  Adjust your join queries to always utilize indexes
  When making adjustments, make tmp_table_size/max_heap_table_size equal
  Reduce your SELECT DISTINCT queries without LIMIT clauses
  Set thread_cache_size to 4 as a starting value
Variables to adjust:
  query_cache_size (>= 8M)
  join_buffer_size (> 128.0K, or always use indexes with joins)
  tmp_table_size (> 16M)
  max_heap_table_size (> 16M)
  thread_cache_size (start at 4)

```

Следует отметить, что MySQLTuner выдаст более эффективные рекомендации, если получит доступ к данным протоколирования, сохраненным на протяжении как минимум одних суток. Тем самым этот скрипт следует запустить спустя как минимум 24 часа после последнего перезапуска MySQL. Выполнив выданные рекомендации, снова запустите скрипт спустя 24 часа. Продолжайте так делать еще несколько дней, с каждым разом получая все более оптимальные настройки MySQL для вашей конфигурации.

Важный источник процессов, ведущих к увеличению времени загрузки, — неоптимизированные, излишние и плохо работающие запросы к MySQL. Выявив и оптимизировав эти медленные SQL-запросы, вы сможете несколько ускорить свой сайт. Хотя проблему медленных запросов в какой-то мере решает кэширование запросов к базе данных, либо на уровне базы данных, либо за счет транзиентов для определенных запросов, при этом никогда не помешает как можно больше оптимизировать исходный SQL-код.

Первое, что нужно сделать для оптимизации запросов к базе данных, — выявить медленные или плохо работающие запросы. С этой задачей прекрасно справляется созданный Джоном Блэкберном плагин Query Monitor (oreil.ly/Glupi).

Плагин Query Monitor, работу с которым иллюстрирует рис. 14.10, добавляет в нижней части страницы панель, позволяющую вам просматривать данные о времени загрузки страницы в миллисекундах, количестве выполненных SQL-запросов и времени их выполнения, сообщения об ошибках и предупреждения в отношении PHP-кода, а также другую полезную информацию.

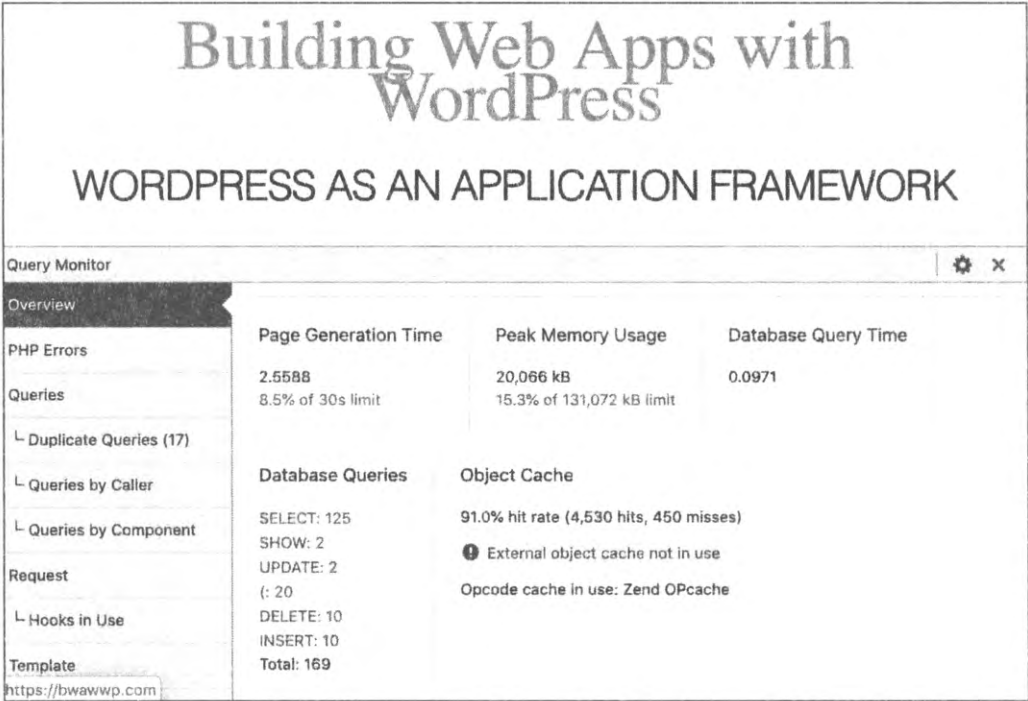


Рис. 14.10. После активации плагина Query Monitor его панель отображается в нижней части всех страниц вашего сайта

Открыв в панели плагина Query Monitor вкладку **Queries** (Запросы), можно просмотреть сведения обо всех выполненных SQL-запросах и времени выполнения каждого отдельного запроса.



Увидеть сгенерированный SQL-запрос в его окончательном виде особенно полезно в случае запросов, составляемых в PHP-коде с помощью нескольких функций или множества логических конструкций. Так, например, окончательный запрос на загрузку постов на главной странице WordPress-блога генерируется с помощью множества переменных, хранящихся в объекте `$wp_query`, в зависимости от того, выполняете ли вы поиск, на какой странице архива вы находитесь, и т. д.

Активировав плагин Query Monitor, вы можете пройтись по страницам своего сайта и найти на них медленно выполняемые или ненужные запросы.

Еще один способ выявления медленных SQL-запросов сводится к тому, чтобы активировать протоколирование медленных запросов в файле конфигурации MySQL. Это позволит вам выявлять медленные запросы в ходе реальной эксплуатации. Протоколирование медленных запросов не является абсолютно необходимым, но иногда позволяет выявить некоторые реальные сценарии использования, которые не проявляют себя на этапе тестирования.

Чтобы активировать протоколирование медленных запросов в MySQL, найдите свой файл `my.cnf` или `my.ini` и добавьте в него следующие строки:

```
slow-query-log = 1;  
slow-query-log-file = /path/to/a/log/file;
```

После внесения изменений в файл конфигурации MySQL потребуется перезапустить MySQL. При оптимизации запросов к базе данных старайтесь выявить следующие ситуации:

- ◆ Случай, когда в ходе загрузки страницы несколько раз выполняется один и тот же SQL-запрос. Сохраните результат этого запроса в глобальной переменной или каким-либо иным образом, чтобы выполнять доступ к нему на последующих этапах загрузки страницы.
- ◆ Случай, когда несколько SQL-запросов можно заменить одним SQL-запросом. Например, плагины могут загружать все свои опции за один раз, вместо того чтобы использовать отдельные запросы или вызовы функции `getOption()` для каждой опции.
- ◆ Случай, когда не используется результат выполняемого SQL-запроса. Некоторые запросы могут оказаться необходимыми только в панели администратора, только в клиентской части или только на определенной странице. Внесите изменения в способ выполнения хука WordPress или снабдите эти вызовы логическими конструкциями языка PHP, чтобы они выполнялись только там, где нужно.

Если вы увидите, что выполнение некоторого необходимого запроса занимает слишком много времени, оптимизируйте его подходящим методом. В частности, можно попытаться выполнить следующее:

- ◆ Скорректируйте запросы так, чтобы в операторах `WHERE`, `ON`, `ORDER BY` и `GROUP BY` использовались только индексируемые столбцы.
- ◆ Добавьте операторы `WHERE` к операторам `JOIN`, чтобы они объединяли подтаблицы меньшего размера.

- ◆ Сохраняйте данные в таблице другого типа: например, заменяйте метаданные постов таксономиями (см. главу 5).
- ◆ Снабдите индексами столбцы, которые необходимы в операторах WHERE, ON, ORDER BY и GROUP BY.

Файлы `advanced-cache.php` и `object-cache.php`

Все описанные методы кэширования, включая те, которые реализованы в плагине W3 Total Cache, опираются на два "краеугольных камня"² — файлы `advanced-cache.php` и `object-cache.php`, добавляемые в каталог `/wp-content/`.

Чтобы дать системе WordPress указание проверять наличие файлов `advanced-cache.php` и `object-cache.php`, добавьте строку `define('WP_CACHE', true);` в файл `wp-config.php`.

Файл `advanced-cache.php` загружается скриптом `wp-settings.php` до загрузки значительной части исходных файлов WordPress. Благодаря этому вы можете выполнить определенный код (например, осуществляющий поиск файла кэша на сервере), а затем прекратить выполнение PHP-кода с помощью команды `exit;` до загрузки остальной части WordPress.

При наличии файла `object-cache.php` он служит для определения функций WP Cache API вместо встроенных функций, определенных в файле `wp-includes/cache.php`. По умолчанию во время загрузки каждой страницы WordPress кэширует все опции в массиве, а транзисты сохраняются в базе данных. Написав собственный файл `object-cache.php`, вы можете дать системе WordPress указание сохранять опции и транзисты в некоторой области оперативной памяти, не освобождаемой при загрузке новых страниц.

Такие плагины, как W3 Total Cache, фактически, являются клиентским модулем для генерирования файла `advanced-cache.php` с учетом заданных вами настроек. Вы также можете создать собственный файл `advanced-cache.php` или `object-cache.php`, или взять файл, уже настроенный на определенный инструмент или метод кэширования. Большинство описываемых далее методов кэширования подразумевает использование специального файла `advanced-cache.php` или `object-cache.php` для взаимодействия с другим сервисом кэширования.

Если в начале своих файлов `*.php`, вставляемых в каталог `wp-content`, вы добавите заголовочный комментарий с такой же, как у плагина, структурой (имя плагина, описание и т.д.), то эта информация будет отображаться в панели администратора WordPress на вкладке **Drop-Ins** (Вставки) страницы плагинов.

Alternative PHP Cache (APC)

Alternative PHP Cache — это расширение для PHP, которое выступает в роли кэша операционных кодов и позволяет кэшировать объекты путем сохранения пар "ключ–значение".

² Конечно, мы знаем, что у здания может быть только один краеугольный камень. Простите нам эту небольшую литературную вольность.

Выполняемый PHP-скрипт компилируется в опкоды, готовые к выполнению сервером. При наличии кэша опкодов часть результатов компиляции кэшируется до тех пор, пока не будут модифицированы исходные PHP-скрипты.

Кэш пар "ключ–значение"

APC также предоставляет функции `apc_store()` и `apc_fetch()`, позволяющие сохранять и извлекать биты информации из оперативной памяти. Значение, записанное в оперативной памяти, как правило, загружается быстрее, чем значение, сохраненное на жестком диске или в базе данных, особенно если нужно не только его извлечь, но и выполнить некоторые вычисления. Такие плагины, как W3 Total Cache или APC Object Cache Backend (oreil.ly/BmMjW), позволяют сохранять объектный кэш WordPress в оперативной памяти с помощью APC.

Последовательность настройки APC выглядит примерно так:

- ◆ Установите APC на сервере, настройте интерпретатор PHP на использование APC и перезапустите веб-сервер.
- ◆ Настройте WordPress на применение APC с помощью плагина W3 Total Cache, плагина APC Object Cache Backend, какого-либо другого плагина или кастомного скрипта `object-cache.php`.

Хорошее описание работы с APC в целом и в сочетании с WordPress можно найти в следующих источниках информации:

- ◆ Раздел "Alternative PHP Cache" сайта PHP.net (bwawwp.com/php-apc-manual).
- ◆ Статья Дэнни Сипоса "Как установить Alternative PHP Cache (APC) на облачном сервере с Ubuntu 12.04" (bwawwp.com/install-php-cache).



В PHP 5.5 и выше предусмотрена компиляция кодов с помощью расширения OP-Cache — альтернативы для APC в плане кэширования опкодов. Однако у OPCache нет таких же функций сохранения и извлечения, какие предлагает APC для объектного кэширования. Из-за этого необходимо либо отключить OPCache и использовать APC, либо запустить в дополнение к OPCache модифицированную версию APC с названием APCu. APCu предлагает функции сохранения и извлечения, оставляя задачу кэширования опкодов расширению OPCache.

Memcached

Memcached — это система, которая позволяет сохранять пары "ключ–значение" в оперативной памяти и может служить в качестве серверного модуля для объектного кэширования в WordPress. Memcached имеет много общего с APC, но не обеспечивает кэширование опкодов.

Сохраняя полностраничные кэши в Memcached, а не в файлах на сервере, можно несколько ускорить загрузку страниц, однако в случае современных серверов с быстрыми твердотельными накопителями этот прирост производительности будет не слишком большим. Memcached сочетается и с сервером Apache, и с сервером Nginx.

Одним из преимуществ Memcached по сравнению с другими методами объектного кэширования (включая Redis и APC) является то, что кэш системы Memcached может быть распределен между несколькими серверами. Поэтому при размещении приложения на нескольких серверах можно будет создать для них общий кэш с помощью одного экземпляра Memcached, вместо того чтобы создавать отдельную (часто избыточную) кэш-память на каждом сервере. Интересно отметить, что корпоративную версию плагина W3 Total Cache можно без проблем использовать в рамках нескольких серверов и APC.

Последовательность настройки Memcached выглядит примерно так:

- ◆ Установите систему Memcached на своем сервере, выделите ей некоторый объем памяти и запустите ее.
- ◆ С помощью плагина W3 Total Cache или Memcached Object Cache (bwawwp.com/wp-memcached) настройте объектный кэш WordPress на использование системы Memcached.

Хорошее описание работы с системой Memcached в целом и в сочетании с WordPress можно найти в следующих источниках информации:

- ◆ Раздел "Memcached" сайта PHP.net (bwawwp.com/php-mem).
- ◆ Сайт системы Memcached (memcached.org/).
- ◆ Статья Скотта Тейлора "WordPress + Memcached" (bwawwp.com/scott-taylor).

Redis

Redis — это еще одна система для сохранения пар "ключ–значение" в оперативной памяти веб-сервера Apache или Nginx. Как и Memcached, она может функционировать в качестве серверного модуля для объектного или страничного кэширования в WordPress.

В отличие от Memcached, Redis позволяет сохранять данные не только в простых хэш-таблицах с парами "ключ–значение", но и в списках, множествах и упорядоченных множествах. Эти структуры данных будут полезны в любом приложении, однако некоторые разработчики высоко ценят зрелость системы Memcached, созданной на несколько лет раньше Redis.

Последовательность настройки Redis выглядит примерно так:

- ◆ Установите систему Redis на своем сервере, выделите ей некоторый объем памяти и запустите ее.
- ◆ Замените скрипт `index.php` системы WordPress каким-либо решением, осуществляющим поиск в кэше Redis и извлекающим оттуда найденные страницы. Для этой цели подойдет плагин WP Redis Cache.
- ◆ С помощью некоторого плагина или скрипта очищайте кэш Redis, например при изменении содержимого постов.

Хорошее описание работы с системой Redis в целом и в сочетании с WordPress можно найти в следующих источниках информации:

- ◆ Сайт системы Redis (redis.io).
- ◆ Описание плагина WP Redis Cache (bawawp.com/wp-redis-cache).
- ◆ Статья Джима Вестергрена "WordPress с Redis в качестве клиентского кэша" (bawawp.com/wp-with-redis).

Varnish

Varnish — это обратный прокси-сервер, который можно разместить перед сервером Apache или Nginx и использовать для доставки посетителям кэшированных версий полных веб-страниц. Поскольку при наличии кэшированных страниц Varnish даже не требует загрузки веб-сервера или интерпретатора PHP, он превосходит Memcached и Redis по эффективности полностраничного кэширования. С другой стороны, Varnish не предназначен для выполнения объектного кэширования и в силу этого позволяет кэшировать только статические страницы сайта.

Последовательность настройки прокси-сервера Varnish для WordPress выглядит примерно так:

- ◆ Установите прокси-сервер Varnish на своем сервере.
- ◆ Настройте прокси-сервер Varnish так, чтобы он игнорировал адрес панели администратора /wp-admin/ и другие разделы сайта, которые не должны подвергаться кэшированию.
- ◆ С помощью плагина или скрипта выполняйте очистку кэша Varnish при изменении содержимого постов и внесении иных изменений на WordPress-сайте. Для этой цели подойдут плагины WP-Varnish (bawawp.com/wp-varnish) и Varnish HTTP Purge (bawawp.com/varnish-http).

Хорошее описание работы с прокси-сервером Varnish в целом и в сочетании с WordPress можно найти в следующих источниках информации:

- ◆ Сайт прокси-сервера Varnish (<https://www.varnish-cache.org/>).
- ◆ Статья Остина Гюнтера "Как установить и настроить Varnish для WordPress" (bawawp.com/custom-varnish).
- ◆ Шаблоны конфигураций Varnish 3.0 (<http://bawawp.com/varnish30>).

Batcache

Плагин Batcache использует APC или Memcached в качестве серверного модуля для полностраничного кэширования в WordPress. Он обеспечивает почти такой же конечный результат, как при полностраничном кэшировании с помощью плагина W3 Total Cache или другого плагина, интегрированного с APC или Memcached.

Уникальная особенность плагина Batcache состоит в том, что он активирует кэширование лишь в том случае, если страница загружается не менее двух раз в течение 120 секунд. В таком случае в кэш заносится копия страницы, которая используется

в течение следующих 300 секунд. Конечно, эти значения можно изменить так, чтобы они больше подходили для вашего случая, но, как вы уже поняли, данный плагин предназначен главным образом для защиты от резких всплесков трафика, вызванных размещением ссылок на ваш сайт в лентах новостей, в популярных блогах и на известных новостных сайтах. Дополнительное преимущество кэширования только высоконагруженных страниц — для хранения такого кэша требуется меньший объем оперативной памяти. При желании также можно изменить исходные параметры плагина Batcache так, чтобы он работал как постоянно активная система полностраничного кэширования.

Последовательность настройки плагина Batcache для WordPress выглядит примерно так:

- ◆ Настройте Memcached или APC для их использования плагином Batcache в качестве размещаемого в оперативной памяти хранилища пар "ключ–значение".
- ◆ Скачайте плагин Batcache из репозитория системы WordPress.
- ◆ Переместите файл `advanced-cache.php` в папку `wp-content` вашего экземпляра WordPress.

Интересным фактом является то, что плагин Batcache был разработан специально для WordPress, и изначально предназначался для сайтов на хостингах WordPress VIP и WordPress.com. Его исходное название — SuperCache, но поскольку примерно в то же время появился популярный плагин WP Super Cache, его создатели заменили имя одного героя комиксов (Супермена) на имя другого (Бэтмена). Более подробные сведения о плагине Batcache и его сочетании с WordPress можно найти в следующих источниках информации:

- ◆ Описание плагина Batcache (bwwwp.com/wp-batcache).
- ◆ Статья Джонатана Д. Джонсона "Кэширование в WordPress с использованием APC и Batcache" (bwwwp.com/wp-batcache).
- ◆ Исходный анонс выхода и обзор Batcache от Энди Скелтона (bwwwp.com/batcache-app).

Выборочное кэширование

Рассматриваемые до сих пор методы кэширования не отличались особой "интеллектуальностью", сохраняя в кэше полные страницы или буквально все объекты WordPress. Хотя их можно дополнить правилами, дающими указание кэшу избегать определенных URL-адресов или условий, обычно они кэшируют все подряд.

В некоторых случаях вам потребуется другой подход, подразумевающий кэширование только определенных страниц и объектов. Обычно это реализуется путем сохранения информации в *транзiente* системы WordPress. Активировав некоторый персистентный объект наподобие APC, можно существенно ускорить загрузку сохраняемой информации.



То, что мы обозначаем здесь *выборочным кэшированием*, часто также называют *фрагментарным кэшированием*. Однако оба названия подразумевают одно и то же: кэширование некоторых частей отображаемой веб-страницы вместо того, чтобы кэшировать ее целиком.

Например, при активации полностраничного кэширования с помощью плагина W3 Total Cache или прокси-сервера Varnish обычно требуется исключить из кэша страницы авторизованных пользователей, чтобы не допустить кэширование информации, относящейся к конкретным членам. В противном случае Мэри может увидеть в правом верхнем углу страницы сообщение "Добро пожаловать, Боб!". Однако в такой ситуации некоторая часть каждой страницы может быть одинаковой у всех пользователей или у определенных категорий пользователей. Мы можем выборочно кэшировать эту информацию, если для ее загрузки требуется выполнять много вычислений или обращений к базе данных.

Выборочному кэшированию рекомендуется подвергать отчеты, сложные запросы постов и другие элементы контента, для обработки которых требуется много времени или значительный объем памяти.

API для работы с транзидентами

Транзистенты — предпочтительный способ внесения и извлечения значений из объектного кэша WordPress-приложений. Если не будет установлена персистентная система кэширования, то транзистенты будут сохраняться в таблице `wp_options` базы данных WordPress. Если же вы установите некоторую систему объектного кэширования, например APC, APCu, Memcached или Redis, то сохранять транзистенты будет эта система.

Сохраненные транзистенты могут быть в любой момент очищены при таком событии, как перезагрузка сервера или очистка памяти, содержащей объектный кэш. Из-за этого при использовании транзистентов всегда следует учитывать, что это временное и ненадежное хранилище. Чтобы исключить потерю определенной информации, сохраните ее в транзистенте для повышения производительности, но также продублируйте ее с помощью какого-либо иного способа — обычно это делается путем сохранения опции с помощью функции `update_option()`.

В приложении SchoolPress нам нужно вычислять для каждого студента средний балл по всем домашним заданиям, которые он сдал. Соответствующий запрос будет включать в себя запуск функции усреднения для столбца `meta_value` таблицы `wp_usermeta`. Вычисление среднего балла не окажет большой нагрузки в случае одного студента, сдавшего, скажем, от 10 до 100 заданий, однако для страницы, отображающей средние баллы 20–80 студентов одного класса, эти вычисления могут занять достаточно значительное время. Чтобы ускорить этот процесс, мы можем кэшировать в транзистенте результаты отчета по всему классу, как показано в листинге 14.1.

Данный случай прекрасно подходит для использования транзистентов, поскольку, имея доступ к сохраненным в транзистенте результатам вычислений, мы сможем ус-

корить повторные загрузки отчета, а случайное удаление транзientа не вызовет тяжелых последствий, так как мы всегда сможем вычислить средние баллы заново.

Листинг 14.1. Класс SPClass

```
class SPClass()
{
    /* ... Конструктор и другие методы ... */

    function getStudents()
    {
        /* получает всех пользователей, входящих в группу BuddyPress класса */

        return $this->students; // массив объектов студентов
    }

    function getAssignmentAverages()
    {
        //проверяем наличие транзientа
        $this->assignment_averages =
            get_transient('class_assignment_averages_' . $this->ID);

        //транзient не найден? вычисляем средние баллы
        if(empty($this->assignment_averages))
        {
            $this->assignment_averages = array();
            $this->getStudents();

            foreach($this->students as $student)
            {
                $this->assignment_averages[$student->ID] =
                    $student->getAssignmentAverages();
            }

            //сохраняем в транзiente
            set_transient('class_assignment_averages_' .
                $this->ID, $this->assignment_averages);
        }

        //возвращаем средние баллы
        return $this->assignment_averages;
    }
}

//очищаем транзientы для средних баллов по заданиям
//в случае выставления оценки за некоторое задание
public function clear_assignment_averages_transient($assignment_id)
```

```
{
//сохраняем идентификатор класса как метаданные поста в посте задания
    $assignment = new Assignment($assignment_id);
    $class_id = $assignment->class_id;

//очищаем транзиент для средних баллов по заданиям, созданный для этого класса
    delete_transient('class_assignment_averages_' . $class_id);
}
add_action('sp_update_assignment_score', array('SPClass',
    'clear_assignment_averages_transient'));
}
```

Мы опустили в данном примере значительную часть кода и сделали ряд допущений в отношении классов `SPClass` и `Student`. Однако из него вполне можно понять, как данный отчет сохраняет вычисленные средние баллы в транзиентах, извлекает эти данные и очищает их в случае обновлений.

В данном примере мы сохраняем в транзиенте массив `$this->assignment_averages`. Мы также могли бы сохранять в транзиенте сгенерированный HTML-код, но сохранение массива позволяет избавиться от большей части сложных вызовов базы данных и обеспечивает большую гибкость.

Для сохранения значения в транзиенте предусмотрена функция `set_transient($transient, $value, $expiration)`, принимающая следующие атрибуты:

- ◆ `$transient` — уникальное имя транзиента длиной до 45 символов;
- ◆ `$value` — сохраняемое значение. Объекты и массивы автоматически сериализуются и десериализуются;
- ◆ `$expiration` — опциональный параметр, задающий срок действия транзиента в секундах. Транзиенты с истекшим сроком действия удаляются скриптом для сбора "мусора" системы WordPress. По умолчанию задано значение 0, при котором срок действия не истекает вплоть до удаления транзиентов.

Обратите внимание, что мы используем описательный ключ (`class_assignment_averages_`), за которым следует идентификатор группы класса. При таком подходе у каждого класса будет собственный транзиент для сохранения средних баллов по заданиям.

Для извлечения содержимого транзиента мы просто вызываем функцию `get_transient($transient)`, передавая ей в качестве параметра уникальное имя транзиента. Если такой транзиент существует и срок его действия еще не истек, возвращается его значение. В противном случае возвращается значение `false`.

Для удаления транзиента до истечения срока его действия мы вызываем функцию `delete_transient($transien)`, передавая ей в качестве параметра уникальное имя транзиента. Обратите внимание, что в данном примере мы подключаемся к хуку `sp_update_assignment_score`, который срабатывает при выставлении оценки за любое задание. В качестве функции обратного вызова для хука передается массив, обеспечивающий вызов метода класса `SPClass`. Хук `sp_update_assignment_score` передает в качестве параметра идентификатор задания `$assignment_id`. Метод обратного

вызова использует этот идентификатор для поиска задания и соответствующего идентификатора класса, после чего удаляет соответствующий транзient `class_assignment_averages_{ID}`.

Если бы мы сохраняли транзиенты для отдельных постов или пользователей, то их можно было бы очищать, соответственно, при сохранении поста (с помощью хука `save_post`) или при обновлении профиля (с помощью хука `profile_update`).



Функции для работы с транзиентами представляют собой просто обертки для трех функций, определенных в исходном файле `wp-includes/cache.php` или добавленном вами файле `object-cache.php`: `wp_cache_set()`, `wp_cache_get()` и `wp_cache_delete()`. При желании эти функции можно вызывать напрямую. Подробное описание этих функций можно найти в Кодексе WordPress (bwwwp.com/class-ref).

Транзиенты в многосайтовом режиме

При сетевой конфигурации транзиенты, определенные с помощью функции `set_transient()`, будут относиться к текущему сайту сети. В нашем случае транзient `class_assignment_averages_1`, определенный на одном сайте сети, не будет доступен на другом сайте сети. (Для ситуации с баллами по заданиям такое поведение будет вполне уместным.)

Для случая, когда требуется определить транзient в рамках всей сети, WordPress предлагает следующие разновидности функций для работы с транзиентами:

```
set_site_transient($stransient, $value, $expiration)
get_site_transient($stransient)
delete_site_transient($stransient)
```

Эти функции работают так же, как и базовые функции для работы с транзиентами, но сохраняют транзиенты не в отдельных таблицах `wp_options` каждого сайта сети, а в таблице `wp_site_options`.

Поскольку при определении транзиентов с помощью функции `set_site_transient()` к их имени добавляется префикс `_site`, максимальная длина имени такого транзиента составляет не 45, а 40 символов.

Наконец, обратите внимание, что до и после определения общесетевого транзиента срабатывает другой набор хуков по сравнению с транзиентом, относящимся к отдельному сайту сети. Если вы привяжете некоторый код к хуку `pre_set_transient`, `set_transient` или `setted_transient`, то его, вероятно, также потребуется привязать к хуку `pre_set_site_transient`, `set_site_transient` и `setted_site_transient`.

Повышение производительности с помощью JavaScript-кода

Для ускорения загрузки страниц часто полезно загружать определенные части веб-страниц с помощью JavaScript-кода, вместо того чтобы генерировать тот же вывод динамически с помощью PHP-кода.



Если ваше приложение изначально создавалось как JavaScript-приложение с применением WP REST API и такого фреймворка, как REACT, то данная мера будет излишней, поскольку вы уже и так загружаете содержимое веб-страниц и экранов приложения с помощью Ajax-запросов.

Такой подход создает впечатление ускорения загрузки страниц за счет того, что сначала быстро загружается каркас страницы, а затем выполняется более длительная загрузка остальной части страницы с отображением мигающего значка загрузки или постепенно заполняющейся полосы загрузки. Пользователи сразу же видят, что страница загрузилась, и понимают, что им нужно подождать еще нескольких секунд, пока отобразится ее содержимое.

Применение JavaScript-кода также может обеспечить и реальное ускорение загрузки страницы. Если весь динамический контент страницы будет загружаться с помощью JavaScript-кода, то остальную часть страницы можно будет загружать из страничного кэша, совсем не используя PHP-код.

Например, у многих блогов единственным динамическим элементом контента являются комментарии. При использовании встроенных комментариев WordPress и полностраничного кэша новые комментарии будут отображаться на сайте только после очистки кэша. Однако если вы воспользуетесь системой комментирования на базе JavaScript, какую, например, предоставляет плагин Jetpack или такие сервисы, как Disqus и Facebook, то ваш раздел комментариев будет представлять собой просто фрагмент статического JavaScript-кода, загружающий динамические комментарии с другого сервера.

Как будет выглядеть загрузка динамической части контента с помощью JavaScript-кода в самом простейшем случае, показано в листинге 14.2.

Листинг 14.2. Плагин JS Display Name

```
<?php
/*
Plugin Name: JS Display Name
Plugin URI: http://bwawwp.com/js-display-name/
Description: Способ загрузки отображаемого имени авторизованного пользователя с помощью JS
Version: .1
Author: Jason Coleman
Author URI: http://bwawwp.com
*/

/*
// используйте эту функцию для размещения JavaScript-кода в своей теме
if(function_exists("jsdn_show_display_name"))
{
    jsdn_show_display_name();
}
```

```

*/
function jsdn_show_display_name($prefix = "Welcome,&nbsp;")
{
?>
<p>
    <script src="<?php echo admin_url(
        "/admin-ajax.php?action=jsdn_show_display_name&prefix=" .
        urlencode($prefix)
    );?>"></script>
</p>
<?php
}

/*
Эта функция выявляет JavaScript-вызов и возвращает отображаемое имя пользователя
*/
function jsdn_wp_ajax()
{
    global $current_user;
    if(!empty($current_user->display_name))
    {
        $prefix = sanitize_text_field($_REQUEST['prefix']);
        $text = $prefix . $current_user->display_name;

        header('Content-Type: text/javascript');
        ?>
        document.write(<?php echo json_encode($text);?>);
        <?php
    }

    exit;
}
add_action('wp_ajax_jsdn_show_display_name', 'jsdn_wp_ajax');
add_action('wp_ajax_nopriv_jsdn_show_display_name', 'jsdn_wp_ajax');

```

Кастомные таблицы

При разработке WordPress-приложений в целом и при оптимизации производительности в частности часто бывает полезно создать кастомную таблицу или представление таблицы базы данных, чтобы ускорить выполнение определенных запросов.

Так, в случае приложения SchoolPress нам потребуется выполнять множество запросов касательно объектов задания. Нам нужно будет сортировать их по баллам,

классам, учителям, студентам, дате выдачи и дате сдачи задания, а также, возможно, по некоторой комбинации этих параметров. При сохранении этих данных в таблице `wp_postmeta` соответствующие запросы будут выполняться медленно, поскольку таблица `wp_postmeta` окажется большой, включая в себя метаданные всех постов, а не только постов заданий, и столбец `meta_value` будет неиндексированным.

Индексация столбца `meta_value` будет избыточной мерой, поскольку при этом будет индексировано много метаданных постов, в индексации которых нет необходимости. Вставка записей в таблицу `wp_postmeta` будет выполняться очень медленно и с большими затратами памяти. Нецелесообразно и преобразование части метаданных постов в таксономии, поскольку этими таксономиями будет сложно управлять, и они вряд ли обеспечат нам необходимое увеличение скорости.

Хотя представленный далее пример является несколько "притянутым за уши", в некоторых случаях будет действительно полезно сохранять данные в кастомной таблице, вместо комбинации постов, метаданных постов и таксономий.

Вот пример таблицы для наших заданий.

Пример

```
CREATE TABLE `wp_sp_assignments` (  
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,  
  `post_id` bigint(20) unsigned,  
  `class_id` bigint(20) unsigned,  
  `student_id` bigint(20) unsigned,  
  `score` int(10),  
  `assignment_date` DATETIME,  
  `due_date` DATETIME,  
  `submission_date` DATETIME  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `post_id` (`post_id`),  
  KEY `class_id` (`class_id`),  
  KEY `student_id` (`student_id`),  
  KEY `score` (`score`),  
  KEY `assignment_date` (`assignment_date`),  
  KEY `due_date` (`due_date`),  
  KEY `submission_date` (`submission_date`)  
);
```

Этот пример представляет собой некоторую крайность — буквально каждый столбец таблицы снабжен индексом. Возможно, такой подход будет избыточным в данном случае, но он позволяет чрезвычайно быстро выполнять запросы к этой таблице, соединенной с таблицей `wp_posts` или `wp_users`.

Создав такую таблицу `wp_sp_assignments`, мы могли бы обновлять ее строки через хук `save_post`, как показано в следующем примере.

```

function sp_update_assignments_table($post_id)
{
    //получаем пост
    $post = get_post($post_id);

    //нам нужны только задания
    if($post->post_type != "assignment")
        return false;

    //готовим данные к вставке или замене
    $assignment_data = array(
        "post_id" => $post_id,
        "student_id" => $post->post_author,
        "teacher_id" => $post->teacher_id,
        "score" => $post->score,
        "assignment_date" => $post->assignment_date,
        "due_date" => $post->due_date
        "submission_date" => $post->submission_date
    );

    //проверяем наличие задания
    $assignment_id = $wpdb->get_var("SELECT id
                                    FROM wp_sp_assignments
                                    WHERE post_id = ' ' . $post_id . ' '
                                    LIMIT 1");

    //если нет задания с указанным идентификатором, создаем новое задание
    if(empty($assignment_id))
    {
        $assignment_id = $wpdb->insert("wp_sp_assignments", $assignment_data);
    }
    else
    {
        $assignment_data['id'] = $assignment_id;
        $wpdb->replace("wp_sp_assignments", $assignment_data);
    }

    return $assignment_id;
}
add_action('save_post', 'sp_update_assignments_table');

```

Действие в обход WordPress

Наконец, еще один метод ускорения загрузки, часто упрощающий масштабирование WordPress-приложения, состоит в том, чтобы вспомнить, что вам не обязательно использовать WordPress для всех элементов приложения.

На самом деле мы уже рассмотрели в этой главе несколько способов применения этой рекомендации. Так, Varnish загружает статические HTML-файлы в обход WordPress. При наличии файла `advanced-cache.php` вы также действуете в обход определенной части WordPress. При загрузке комментариев из социальной сети Facebook с помощью JavaScript-кода вы тоже действуете в обход WordPress. Когда вы сохраняете часть данных в кастомной таблице, вы опять же действуете в обход инфраструктуры WordPress.

Мы используем WordPress для создания приложений по той причине, что это дает нам такие преимущества, как безопасность, функциональность, множество доступных плагинов и решений. Создавая свой код на базе WordPress, мы можем легко подключаться к хукам этой системы для управления контентом и пользователями. Мы также можем легко добавлять хуки в собственный код.

Однако иногда недостаток производительности перевешивает все эти преимущества. В таком случае необязательно отказываться от WordPress целиком; достаточно будет обойти лишь определенные функции этой системы.

Например, в приведенном ранее примере скрипта для получения отображаемого имени пользователя вместо JavaScript-кода можно было бы написать PHP-код, который с помощью простого запроса `SELECT` извлекал бы значение столбца `display_name` для пользователя, указанного в созданном системой WordPress cookie-файле. Это позволило бы сэкономить несколько миллисекунд при загрузке каждой страницы сайта. Если суммировать эту экономию для всех динамических элементов приложения, получится достаточно значительный эффект.

В большинстве случаев к подобным действиям в обход WordPress следует прибегать лишь в качестве крайней меры. Хотя это и дает некоторый прирост скорости, за него приходится платить усложнением кода. Такие скрипты лучше запускать с помощью REST API или скрипта `admin-ajax.php`, как мы делали ранее — это избавляет вас от большей части издержек системы WordPress, в то же время позволяя при необходимости взаимодействовать из кода с другими плагинами и задействовать встроенные классы и API системы WordPress.

Если вы загружаете скрипт для экспорта файлов в формате CSV, выполнение которого в любом случае занимает 10 секунд, то экономия в 0,5 секунды не будет иметь никакого значения.

При написании программного кода всегда старайтесь придерживаться как можно более простого подхода, что в данном случае означает применение обычных средств системы WordPress, и оптимизируйте этот уровень лишь в случае выявления здесь узких мест на этапе разработки. При этом следует изучить каждый из представленных в этой главе вариантов и выбрать из них тот, который лучше всего подходит для оптимизируемого вами элемента, исходя из ваших потребностей, состава команды разработчиков и имеющихся в наличии инструментов.

Электронная коммерция

В какой-то момент у вас может возникнуть желание взимать плату за доступ к вашему приложению или принимать какие-нибудь другие платежи на своем сайте. В этой главе мы пройдемся по лучшим плагинам для электронной коммерции и платных подписок и дадим вам несколько подсказок о том, на чем следует основывать свой выбор. Мы также рассмотрим пошаговую инструкцию по настройке типичной системы платного доступа в модели SaaS (Software-as-a-Service — программное обеспечение как услуга).

Выбор плагина

У WordPress есть один плагин, который является чуть ли не синонимом электронной коммерции: WooCommerce. Начиная с 2014 года, когда было опубликовано первое издание этой книги, WooCommerce стал доминирующей платформой в данной области, и не только для WordPress. Здесь мы представим краткое описание этого плагина и рассмотрим несколько хуков и фильтров, которые могут заинтересовать разработчиков приложений.

Несмотря на то, что WooCommerce — отлично написанный и поддерживаемый продукт, в некоторых случаях более уместными могут оказаться и другие плагины. В связи с этим будут рассмотрены Paid Memberships Pro (плагин с акцентом на платные подписки) и Easy Digital Downloads (плагин с акцентом на виртуальные товары).

Все плагины, представленные в этой главе, обладают следующими возможностями:

- ♦ интеграция с несколькими платежными системами;
- ♦ безопасные формы оплаты;
- ♦ сохранение информации о заказе;
- ♦ перечисление продуктов (уровней подписки) с ценами.

Уникальные особенности каждого вида плагинов для электронной коммерции будут освещены в следующих разделах.

WooCommerce

Сложно подсчитать, какую долю рынка занимает платформа электронной коммерции¹, но, по всей видимости, как минимум 20% *всех* сайтов, которые занимаются

¹ Что мы понимаем под "платформой электронной коммерции"? На чем должна быть основана рыночная доля: на количестве сайтов или продаж? Каким образом следует учитывать готовые удаленные решения, такие как Shopify или Amazon?

торговлей, основаны на WooCommerce². Если верить WordPress.org (oreil.ly/odzOF), этот плагин был загружен свыше 47 миллионов раз на более чем 4 миллионах активных сайтов. Как бы вы ни считали, WooCommerce используется на огромном количестве ресурсов. Вот почему компания Automattic купила WooCommerce еще в 2015 году, и именно поэтому данный продукт будет в центре нашего внимания.

WooCommerce, как и другие "корзины покупок", обладает следующими возможностями:

- ◆ пользовательские типы постов для продуктов;
- ◆ навигация по продуктам;
- ◆ возможность искать по продуктам;
- ◆ возможность купить сразу несколько продуктов;
- ◆ вычисление стоимости доставки с возможностью указания адреса;
- ◆ поддержка пользовательских правил для налогов.

Документацию о том, как настроить WooCommerce, и многое другое можно найти на сайте WooCommerce (docs.woocommerce.com/).

Плагин и расширения WooCommerce

Плагин WooCommerce доступен бесплатно в репозитории WordPress.org. Его основная часть содержит все, что необходимо для определения товаров и цен, а также для обработки платежей через PayPal. Недавно в бесплатном доступе появилось расширение Stripe Payment Gateway для WooCommerce, которое является самым популярным средством работы с кредитными картами непосредственно на вашем сайте, но аналогичные расширения существуют практически для любого способа оплаты.

У WooCommerce есть десятки платных и бесплатных расширений, которые улучшают основной плагин, интегрируют его со сторонними рекламными сервисами, добавляют новые типы товаров, предоставляют поддержку дополнительных служб доставки или упрощают управление магазином. Два самых популярных из них — WooCommerce Subscriptions (oreil.ly/PqCwh; позволяет запоминать повторяющиеся платежи) и WooCommerce Memberships (oreil.ly/8lf8C; тоже запоминает повторяющиеся платежи, но также умеет ограничивать доступ к контенту в зависимости от уровня подписки). Специально для этого сценария разработан плагин Paid Memberships Pro, о котором мы поговорим чуть позже, но если вы уже работаете с WooCommerce или вам нужна какая-то другая функция, с которой он хорошо справляется, перечисленные расширения будут хорошим выбором.

Изменение поведения WooCommerce с помощью хуков

WooCommerce (как и любой хороший плагин для WordPress) имеет огромное количество хуков с действиями и фильтрами, с помощью которых можно влиять на то,

² Согласно оценкам сервиса BuiltWith (oreil.ly/sprAP), на момент написания этой книги из миллиона самых популярных сайтов, занимающихся электронной коммерцией, 21% используют WooCommerce.

как он работает на вашем сайте. Их полный список можно найти по адресу oreil.ly/Kk0ie.

Далее приводятся примеры того, что можно сделать с помощью некоторых типичных хуков WooCommerce.

- ◆ *Подготовка распродажи в рамках всего сайта.* Глобальную распродажу в магазине WooCommerce можно организовать и с использованием готовых плагинов, но, поскольку вы не лыком шитый разработчик, вам, возможно, захочется сделать это самостоятельно с помощью самописного кода. В следующем примере устанавливается скидка в размере 10% для любого товара, который не участвует ни в какой другой распродаже.

Пример

```
// Устанавливаем скидку в 10% от обычной цены,
// если товар еще не находится в распродаже
function my_get_sale_price($sale_price, $product) {
    if(empty($sale_price)) {
        $sale_price = $product->get_regular_price() * .9;
        $product->set_price($sale_price);
    }

    return $sale_price;
}

add_filter('woocommerce_product_get_sale_price', 'my_get_sale_price',
    10, 2);
add_filter('woocommerce_product_variation_get_sale_price',
    'my_get_sale_price', 10, 2);
```

В WooCommerce у товаров есть как "обычная", так и "распродажная" цена: `regular_price` и `sale_price`. Итоговая вычисленная цена называется просто `price`. Для установки распродажной цены предусмотрен хук `woocommerce_product_get_sale_price`, но для поддержки нескольких разновидностей товаров (например, футболок размеров S, M и L) вам также понадобится хук `woocommerce_product_get_variation_sale_price`.

Обратите внимание на строки с вызовом `$product->set_price($sale_price)`. Вычисляемая цена не обновляется автоматически после возвращения распродажной цены из этой функции обратного вызова; мы должны делать это вручную. Параметр `$product` передается в функцию по ссылке, поэтому обновление, которое здесь выполняется, изменяет товар и за пределами фильтра.

- ◆ *Автоматическое завершение заказов.* По умолчанию новому заказу, который создается на странице оплаты, присваивается статус "в ожидании", и администратор сайта должен его обработать. Для физических товаров *обработка* обычно подразумевает упаковку и отправку заказанного товара с последующим присвоением заказу статуса "завершен".

Заказы виртуальных товаров имеет смысл автоматически делать "завершенными". Это экономит вам лишний щелчок мыши и позволяет плагину WooCommerce быстрее среагировать на завершение заказа — отправить электронные письма, обратиться к соответствующим API-интерфейсам и т.д. Код следующего примера перебирает товары в корзине после оформления заказа и, если все они виртуальные, автоматически делает заказ завершенным.

Пример

```
function autocomplete_virtual_orders($order_id) {
    // Получаем существующий заказ
    $order = new WC_Order($order_id);

    // подразумевается, что включено автозавершение
    $autocomplete = true;

    // проходимся по заказанным товарам
    if (count($order->get_items()) > 0) {
        foreach ($order->get_items() as $item) {
            if($item['type'] == 'line_item') {
                $_product = $order->get_product_from_item($item);
                if(!$_product->is_virtual()) {
                    // в корзине найден неvirtуальный товар
                    $autocomplete = false;
                    break;
                }
            }
        }
    }

    // при необходимости меняем статус
    if(!empty($autocomplete)) {
        $order->update_status('completed', 'Autocompleted.');
```

Paid Memberships Pro

Paid Memberships Pro и другие подобные плагины предназначены для приема платежей за доступ к приложению или сайту WordPress по подписке. Эти плагины включают в себя такие возможности:

- ◆ регулярная плата за подписку;
- ◆ средства блокирования доступа к контенту в зависимости от уровня подписки.

Документацию о том, как настроить Paid Memberships Pro, и многое другое можно найти на веб-сайте по адресу oreil.ly/mrILX.

За что нам нравится Paid Memberships Pro

Плагин Paid Memberships Pro был разработан соавтором этой книги, Джейсоном Коулманом, но это далеко не единственное его преимущество. Это единственный плагин WordPress для подписок, который полностью распространяется под лицензией GPL (General Public License) и доступен бесплатно в репозитории WordPress. Другие плагины предоставляют часть своих возможностей либо в виде платных модулей, либо в отдельных платных версиях.

Весь код Paid Memberships Pro находится в публичных репозиториях на GitHub, и в его разработке может принимать участие кто угодно. Как и WooCommerce, он поддерживает хуки и фильтры для изменения стандартного поведения.

Почти у любого сайта с платными подписками есть свой собственный способ подсчета стоимости перехода на более высокий уровень подписки, специальных предложений или того, как и когда следует ограничивать доступ к контенту. У Paid Memberships Pro нет длинной страницы настроек; вместо этого данный плагин предлагает тщательно продуманные хуки и фильтры, которые позволяют создать практически любую модель ценообразования или ограничения контента с помощью всего нескольких строчек кода.

Еще одно ключевое отличие Paid Memberships Pro от аналогичных плагинов состоит в наличии собственной таблицы с уровнями подписок и информацией о том, как они связаны с пользователями и заказами. Некоторые плагины используют для определения уровней подписки встроенные в WordPress пользовательские роли. В некоторых сайтах с платными подписками (см. главу 6) пользовательские роли имеют большое значение, но в целом их лучше не привязывать к уровням подписки; таким образом одни зарегистрированные пользователи смогут быть администраторами, а другие подписчиками. Если же вам нужно назначать пользовательские роли в зависимости от уровня подписки, это можно легко сделать и в Paid Memberships Pro. Пример этого будет показан чуть позже в этой главе.

Позже вы сможете познакомиться с несколькими примерами использования Paid Memberships Pro. Но прежде чем переходить к общим концепциям электронной коммерции, давайте рассмотрим еще один уникальный плагин.

Easy Digital Downloads

Все плагины электронной коммерции, которые упоминались до сих пор, пригодны не только для физических изделий, но и для цифровых товаров и загрузок. Если же вы планируете продавать исключительно цифровые товары, вам стоит взглянуть на плагин Easy Digital Downloads (easydigitaldownloads.com), разработанный специально для этого. Он обладает следующими возможностями:

- ♦ загрузка файлов, доступная только авторизованным клиентам;
- ♦ возможность оплачивать сразу несколько загрузок.

Документацию о настройке Easy Digital Downloads и многое другое можно найти на веб-сайте этого продукта (oreil.ly/cpU1C).

За что нам нравится Easy Digital Downloads

Easy Digital Downloads включает в себя расширения, такие как Software Licensing and Product Support, которые могут пригодиться разработчикам. Основной плагин и все его расширения отлично написаны и имеют хорошую поддержку.

Как и WooCommerce, плагин Easy Digital Downloads доступен бесплатно в репозитории WordPress, а его расширения можно купить на официальном веб-сайте как по отдельности, так и с помощью пропусков, которые открывают доступ к большинству или всем расширениям по сниженной цене (что предпочтительней для большинства пользователей).

Примеры кода для работы с Easy Digital Downloads

Далее приведены примеры кода, которые демонстрируют использование функций и хуков Easy Digital Downloads (EDD) в более крупном приложении на основе WordPress. Проверить, купил ли пользователь определенную загрузку EDD, можно с помощью функции `edd_has_purchased()`. Если вы уже применяете EDD для продажи загрузок на своем сайте, эта функция позволит вам ограничивать доступ к другим страницам WordPress или возможностям вашего приложения.

Пример

```
// Ограничиваем доступ к странице, если пользователь еще не купил определенную
загрузку
function my_template_redirect_check_edd()
{
    global $current_user;

    // Задаем слаг защищенной страницы.
    $protected_page_slug = 'customers-only';

    // Задаем ID загрузки, которую нужно проверить.
    $required_download_id = 184;

    // Защищаем только одну конкретную страницу.
    if(!is_page($protected_page_slug))
        return;

    // Перенаправляем, если пользователь не зарегистрировался
    // или не сделал покупку.
    if(!is_user_logged_in() ||
        !edd_has_user_purchased($current_user->ID, $required_download_id)) {
        wp_redirect(get_permalink($required_download_id));
        exit;
    }
}

add_action('template_redirect', 'my_template_redirect_check_edd');
```

В этом примере мы использовали хук `template_redirect` из WordPress. Пользователи, которые не купили проверяемую нами загрузку, перенаправляются на ее клиентскую страницу. Это хороший способ заблокировать доступ ко всей странице (которая, возможно, содержит контент, короткий номер для контактной формы или информацию, доступ к которой должны иметь только ваши клиенты).

Аналогичную проверку можно было бы реализовать в виде обертки вокруг любого вашего кода на PHP. Регулярное применение этого подхода поможет вынести `edd_has_user_purchased()` в отдельную функцию. В следующем примере проверяется, купил ли пользователь определенную загрузку, и затем вызывается вспомогательная функция, которая добавляет ссылку на страницу поддержки в первичное меню.

Пример

```
// Вспомогательная функция, которая проверяет, является ли пользователь клиентом
function is_plugin_customer($user_id = null)
{
    $plugin_download_id = 184;    // обновляем это значение

    // по умолчанию выбираем текущего пользователя
    if(empty($user_id))
    {
        global $current_user;
        $user_id = $current_user->ID;
    }

    return edd_has_user_purchased($user_id, $plugin_download_id);
}

// Добавляем ссылку на страницу поддержки в первичное меню
// (для пользователей, сделавших покупку)
function add_support_link_to_menu($items, $args)
{
    if($args->theme_location == 'primary' && is_plugin_customer())
    {
        $items .= '<li class="menu-item menu-item-type-post_type
                        menu-item-object-page menu-item-support">';
        $items .= '<a href="/support/">Support</a>';
        $items .= '</li>';
    }

    return $items;
}

add_filter('wp_nav_menu_items', 'add_support_link_to_menu', 10, 2);
```

Функция `is_plugin_customer()` обеспечивает удобный прием при работе с параметром `$user_id`. Вы можете передать для проверки определенный ID или оставить параметр пустым, в результате чего функция выберет по умолчанию ID текущего пользователя. ID загрузки плагина (ID поста, который видно при редактировании этой загрузки на панели управления) прописан прямо в функции и используется в возвращаемом вызове `edd_has_user_purchased()`.

В последней строчке кода мы добавили фильтр `wp_nav_menu_items`, чтобы создать дополнительную ссылку на страницу поддержки для пользователей, которые являются клиентами.

Теперь давайте обсудим некоторые общие концепции электронной коммерции, а затем рассмотрим примеры того, как интегрировать сервис SaaS и WordPress.

Платежные системы

Платежная система — это сервис, который обрабатывает и иногда хранит информацию о кредитной карте клиента, заботясь о том, чтобы деньги попали на ваш банковский счет³. В США популярностью пользуются такие платежные системы, как Stripe, PayPal, Authorize.net и Braintree Payments. Таких систем насчитываются десятки, и многие из них специализируются на определенных регионах или рынках.

Вот несколько обстоятельств, о которых стоит подумать при выборе платежной системы:

- ◆ Поддерживает ли эта система вашу страну и валюту, с которой вы работаете?
- ◆ Интегрирована ли эта система в плагин электронной коммерции, который вы используете?
- ◆ Поддерживает ли эта система область предпринимательства, в которой вы работаете? Некоторые системы отказываются принимать платежи на сайтах для взрослых, сайтах с азартными играми и других ресурсах "повышенного риска".
- ◆ Предлагает ли эта система нужные вам возможности, такие как периодическое выставление счета или хранение кредитных карт?
- ◆ Насколько система соответствует стандарту безопасности Payment Card Industry (PCI)?⁴
- ◆ Совместима ли эта система с вашим торговым счетом (подробней об этом в следующем разделе)?
- ◆ Наконец, каков размер комиссии? Один процент от 10\$ миллионов — большая сумма, и поиск системы с более низкими отчислениями стоит потраченного

³ С вычетом всех комиссий.

⁴ В идеале совместимость PCI требует более дорогой серверной конфигурации и отдельных постоянных работников для ее обслуживания и документирования. Некоторые платежные системы предоставляют собственные технологии и процедуры, которые помогут вам избежать лишних расходов и при этом обеспечить безопасное хранение клиентских данных.

времени. Но в целом платежные системы имеют довольно стандартные размеры комиссии, поэтому, прежде всего, нужно обращать внимание на то, совместимы ли они с вашей бизнес-конфигурацией. Более крупным и доходным компаниям легче договориться о снижении комиссии до стандартного минимума, принятого в соответствующей отрасли.

Торговые счета

Торговые счета (или счета продавцов) часто путают с платежными системами, но обычно это отдельная сущность, которая необходима для обработки платежей на вашем веб-сайте. Путаница частично вызвана тем фактом, что некоторые платежные системы имеют собственные торговые счета.

Как бы то ни было, для заработка в Интернете требуется как платежная система, так и торговый счет. Они предоставляются двумя разными видами провайдеров, каждый из которых помогает с регистрацией другого. Иными словами, поиск платежной системы помогает найти торговый счет, а поиск торгового счета помогает найти платежную систему. Как показывает опыт, молодым компаниям легче договориться о выгодной комиссии, если они открывают торговый счет через свою платежную систему, а не в своем банке.

Вот какой путь проходят деньги и информация о кредитной карте, когда клиент делает покупку: WordPress → плагин электронной коммерции → платежная система → торговый счет → ваш расчетный счет.

Одно из различий между платежными системами и торговыми счетами состоит в том, что первые в основном относятся к технологиям, а вторые — к бизнесу. Платежная система предоставляет технологию, с помощью которой вы можете проверить кредитную карту, запросить снятие средств и настроить регулярные платежи. Кроме того, некоторые системы могут хранить информацию о клиентах на случай дальнейших покупок.

Торговым является такой банковский счет, который хранит поступающие средства, пока их нельзя будет перевести на ваш расчетный счет. Почему это не происходит напрямую? Данная задержка аналогична ожиданию в ходе клиринга чека или проверки кредитной карты. Если по какой-либо причине компании, выдавшей кредитную карту, нужно запросить возврат средств (из-за возникшей ошибки или по просьбе клиента), во время этой задержки она может снять деньги с вашего торгового счета.

При выборе торгового счета нужно обращать внимание на следующие важные аспекты:

- ◆ Совместима ли моя платежная система с этим торговым счетом?
- ◆ Подходит ли этот торговый счет для моего вида предпринимательской деятельности? Некоторые торговые счета несовместимы с азартными играми и другими видами бизнеса "повышенного риска".

- ♦ Подходит ли этот торговый счет для бизнеса моего масштаба? Некоторые провайдеры не дадут открыть торговый счет новой компании, которая продает дорогие товары (стоимостью в тысячи долларов).
- ♦ Наконец, какова комиссия? Иногда она взимается в процессе покупки, а иногда ее нужно платить отдельно.

Вначале лучше всего выбрать плагин, затем совместимую с ним платежную систему, и только после этого пытаться зарегистрировать через нее торговый счет.

Настройка модели SaaS с помощью Paid Memberships Pro

Модель платного доступа к веб-приложению называется SaaS (читается как "саас"). В этом разделе мы обсудим то, как сконфигурировать Paid Memberships Pro в нашем приложении SchoolPress, чтобы школы могли оформить платную подписку в размере 1000\$ в год.

Модель SaaS

Модель SaaS по сути означает, что вместо покупки "коробочной" версии программного обеспечения с последующей его установкой на своем компьютере вы платите (обычно ежемесячно или ежегодно) за доступ к облачному веб-приложению. Примерами сервисов, которые используют модель SaaS, являются GitHub, Dropbox, Evernote, Google Apps и даже пакет Microsoft Office, который теперь доступен по подписке.

Популярность модели SaaS объясняется тем, что она генерирует относительно предсказуемый и регулярный доход. Но она также является важным элементом заработка на открытом программном обеспечении, таком как WordPress. Проект WordPress распространяется под лицензией GPL, согласно которой, если вы продаете и распространяете свое ПО, ваши клиенты получают право самостоятельно распространять ваш код — возможно, бесплатно. Поэтому продажа услуг по модели SaaS позволяет вашим клиентам пользоваться вашим ПО, но без права дальнейшего распространения исходного кода.

Если вы хотите получать единовременные, ежемесячные или ежегодные платежи за доступ к вашему приложению, в этом вам помогут следующие инструкции.

Шаг 0: определение способа оплаты

Что вы продаете: пожизненную лицензию или месячную подписку? Возможно, это годовая подписка? Хотите ли вы, чтобы счет выставлялся автоматически каждый год, или клиент сам должен будет продлевать доступ к приложению?

Постарайтесь как можно точнее ответить на эти вопросы, прежде чем интегрировать Paid Memberships Pro или писать свой собственный код. У Джейсона есть хо-

роший цикл статей о том, как устанавливать цену за доступ к своим веб-приложениям и сайтам с премиальным контентом (bwawwp.com/pmp-pricing).

Что касается нашего приложения SchoolPress, мы будем ежегодно выставять каждой школе счет в размере 1000\$. В ходе регистрации мы создадим для школы дочерний домен вида **myschool.schoolpress.com** с сайтом WordPress и выдадим ее представителям администраторский доступ к этому сайту, чтобы они могли добавлять туда своих учителей и другие данные.

Мы сконфигурируем подписку таким образом, чтобы школе раз в год автоматически выставлялся счет.

Шаг 1: установка и активация Paid Memberships Pro

Плагин Paid Memberships Pro доступен в репозитории WordPress, благодаря чему его установка и активация не составляет труда (рис. 15.1).

1. Откройте панель управления WordPress и перейдите в раздел **Plugins** → **Add New** (Плагины → Добавить новый).
2. Найдите в списке Paid Memberships Pro и щелкните по ссылке **Install** (Установить).
3. При необходимости здесь можно ввести информацию о FTP (некоторые хостинги этого не требуют).
4. После успешной установки плагина щелкните по ссылке **Activate** (Активировать).

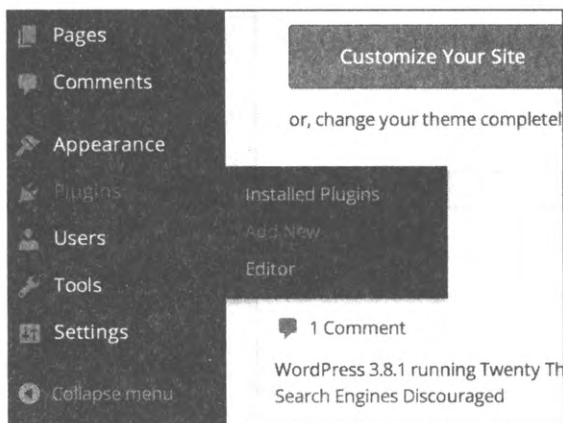


Рис. 15.1. Добавление нового плагина

Шаг 2: задание уровней подписки

Теперь вам нужно сконфигурировать уровни подписки:

1. Откройте панель управления WordPress и перейдите на только что созданную страницу **Memberships** (Подписки).
2. Щелкните по ссылке или кнопке **Add new level** (Добавить новый уровень).

3. Укажите в полях ввода информацию о подписке, как показано на рис. 15.2.

Add New Membership Level

ID:

Name:

Description:

Confirmation Message:

Billing Details

Initial Payment: \$ The initial amount collected at registration.

Recurring Subscription: ☐ Check if this level has a recurring subscription payment.

Other Settings

Disable New Signups: ☐ Check to hide this level from the membership levels page and disable registration.

Membership Expiration: ☐ Check this to set when membership access expires.

Content Settings

Categories: ☐ Uncategorized

Рис. 15.2. Страница добавления нового уровня подписки в Paid Memberships Pro

В нашем случае это будет выглядеть так:

Name (Название): <название школы>.

Description (Описание): здесь должно регистрироваться руководство школы, чтобы получить доступ к вашему сайту SchoolPress.

Confirmation Message (Подтверждающее сообщение): можно не заполнять.

Initial Payment (Начальный платеж): 1000.

Recurring Subscription (Регулярные платежи): установлено.

Billing Amount (Размер платежа): 1000.

Per (Срок): 1.

Days/Weeks/Years (Дней/Недель/Лет): **Years** (Лет).

Billing Cycle Limit (Лимит на количество платежей): 0.

Custom Trial (Пользовательский пробный период): не установлено.

Disable New Signups (Отключить регистрацию новых пользователей): не установлено.

Membership Expiration (Ограниченный срок действия подписки): не установлено.

Categories (Категории): все сброшены.

4. Нажмите кнопку **Save Level** (Сохранить уровень).

Шаг 3: конфигурация страниц

Плагину Paid Memberships Pro нужно несколько страниц, чтобы вместить весь процесс оплаты и другие функции, связанные с подпиской. Если перейти на вкладку **Pages** (Страницы) в настройках Paid Memberships Pro, можно увидеть форму, показанную на рис. 15.3.

Рис. 15.3. Генерация страниц для Paid Memberships Pro

Если у вас уже есть страницы, предназначенные для описания ваших уровней или учетной записи пользователя, вы можете выбрать их в раскрывающихся списках на вкладке **Pages** (Страницы) настроек Paid Memberships Pro. Но в большинстве случаев лучше щелкнуть по ссылке **click here to let us generate them for you** (щелкните здесь, чтобы мы сгенерировали их для вас). Будут созданы такие страницы: **Ac-**

count (Учетная запись), **Billing Information** (Платежные данные), **Cancel** (Отмена), **Checkout** (Оформление заказа), **Confirmation** (Подтверждение), **Invoice** (Счет-фактура) и **Levels** (Уровни). То, как будут выглядеть эти страницы, сгенерированные скриптом, показано на рис. 15.4.

Pages

The following pages have been created for you: 50, 51, 52, 53, 54, 55, 56.

Manage the WordPress pages assigned to each required Paid Memberships Pro page.

Account Page:	Membership Account ▼	edit page	view page
Include the shortcode [pmpo_account].			
Billing Information Page:	Membership Billing ▼	edit page	view page
Include the shortcode [pmpo_billing].			
Cancel Page:	Membership Cancel ▼	edit page	view page
Include the shortcode [pmpo_cancel].			
Checkout Page:	Membership Checkout ▼	edit page	view page
Include the shortcode [pmpo_checkout].			
Confirmation Page:	Membership Confirmation ▼	edit page	view page
Include the shortcode [pmpo_confirmation].			
Invoice Page:	Membership Invoice ▼	edit page	view page
Include the shortcode [pmpo_invoice].			
Levels Page:	Membership Levels ▼	edit page	view page
Include the shortcode [pmpo_levels].			

Рис. 15.4. Страницы, сгенерированные по умолчанию плагином Paid Memberships Pro

Шаг 4: выбор параметров оплаты

На рис. 15.5 показана вкладка **Payment Gateway & SSL** (Платежная система и SSL) в настройках Paid Memberships Pro. Здесь нужно выбрать платежную систему и указать значения для соответствующего пользователя и/или API-интерфейса. В зависимости от выбранной системы на этой странице можно будет поменять валюту, определить доступные кредитные карты, включить/выключить SSL (не забывайте, SSL следует устанавливать везде, кроме тестовых сайтов) и решить, нужно ли выполнить полный сброс SSL-сертификата.

На странице настроек оплаты можно также вставить код SSL-печати и ввести свой штат и размер взимаемого налога. Подсчет налоговых отчислений можно делать программным путем, с помощью фильтра `pmpo_tax` (подробней о нем чуть позже).

PaidMembershipsPro v1.7.6 [Plugin Support](#) [User Forum](#)

Membership Levels Pages **Payment Gateway & SSL** Email Advanced Add Ons

Payment Gateway & SSL Settings

Learn more about [SSL](#) or [Payment Gateway Settings](#).

Payment Gateway: Testing Only

Gateway Environment: Sandbox/Testing

Currency: US Dollars (\$) Not all currencies will be supported by every gateway. Please check

Accepted Credit Card ☒ Visa ☒ Mastercard

Рис. 15.5. Параметры оплаты в Paid Memberships Pro

На странице с настройками оплаты также выводится URL-адрес, который нужно предоставить вашей платежной системе, чтобы она могла взаимодействовать с вашим сайтом. В разных системах эта функция называется по-разному: "IPN handler" в PayPal, "silent post URL" в Authorize.net и "webhook" в Stripe и Braintree.

Шаг 5: выбор настроек электронной почты

По умолчанию WordPress отправляет электронные письма вашего сайта с адреса **wordpress@<ваш_сайт>.com**, который выглядит не самым изящным образом и может попросту не существовать. Вкладка **Email** (Электронная почта) в настройках Paid Memberships Pro, показанная на рис. 15.6, позволяет переопределить эти значения и указать, какие администраторские письма, относящиеся к подпискам, вы хотите получать.

Независимо от того, используете вы Paid Memberships Pro или нет, у вас может возникнуть необходимость в отправке более изящных электронных писем. Чтобы этого добиться, обратитесь к коду по адресу oreil.ly/omUNn, относящемуся к электронной почте.

Шаг 6: Выбор дополнительных настроек

На рис. 15.7 показана вкладка **Advanced Settings** (Дополнительные настройки), которая предоставляет несколько встроенных параметров работы Paid Memberships Pro. Особое внимание стоит обратить на условия предоставления услуг, которые выводятся пользователям при регистрации. Это текстовое поле с прокруткой, в котором отображается страница Terms of Service (TOS), и флажок, который нужно установить, чтобы согласиться с условиями.

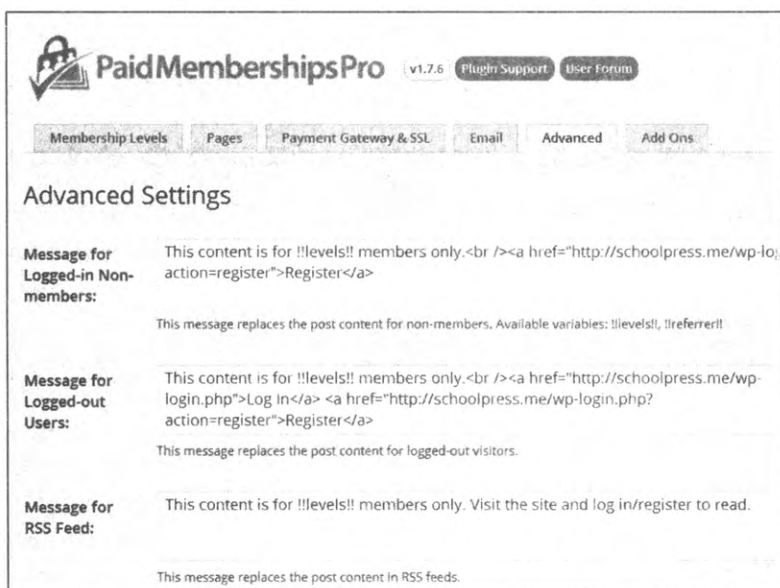


Рис. 15.6. Настройки электронной почты в Paid Memberships Pro

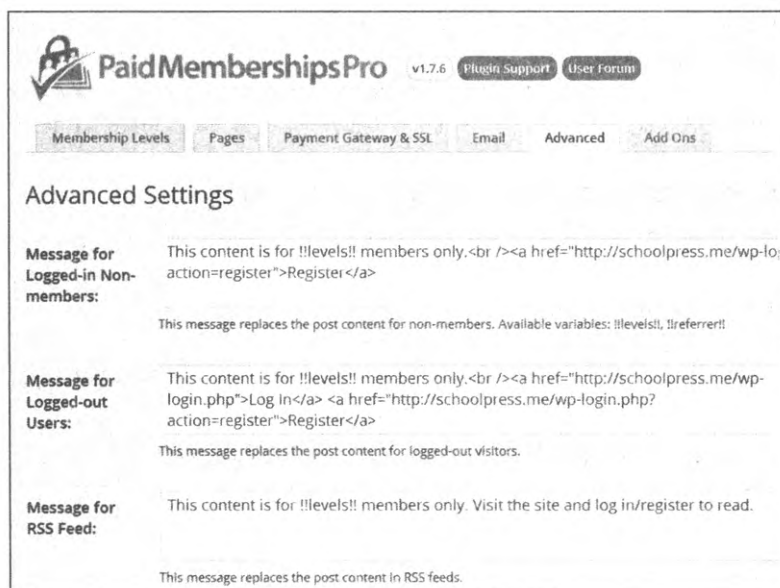


Рис. 15.7. Дополнительные настройки в Paid Memberships Pro

По состоянию на май 2018 года Paid Memberships Pro интегрируется с основной функциональностью WordPress для поддержки Общего регламента по защите данных (General Data Protection Regulation или GDPR), который действует в ЕС. Если у вашего публичного приложения будут пользователи из Евросоюза (или если вас просто заботит безопасность личных данных), вы должны подумать о том, насколько ваш проект соответствует GDPR. В своей статье в блоге Paid Memberships Pro

(oreil.ly/ΠΕ0В) Джейсон описывает свое отношение к GDPR и то, как он добавил его поддержку в свой проект.

Шаг 7: блокирование доступа к страницам

Если не считать генерации страницы оформления заказа и интеграции с вашей платежной системой, главная функциональность, которую привносит Paid Memberships Pro, состоит в возможности блокировать доступ к определенным страницам или отдельным их участкам в зависимости от уровня подписки пользователя. Это можно сделать несколькими разными способами.

- ◆ **Блокирование доступа к определенной странице.** Paid Memberships Pro добавляет на боковую панель редактирования поста и редактирования страницы в настройках WordPress флажок **Require Membership** (Требовать подписку), как показано на рис. 15.8. Чтобы заблокировать доступ к странице для определенных уровней подписки, установите соответствующие флажки.

Если выбрано сразу несколько уровней, *любой* из них позволит просматривать страницу. Если не выбрано ни одного уровня, страница будет доступна для всех (включая незарегистрированных посетителей).

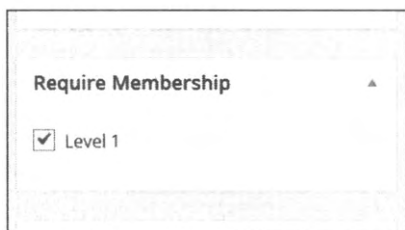


Рис. 15.8. Выбор уровней, необходимых для просмотра страницы

- ◆ **Блокирование страницы по URL-адресу.** Иногда, чтобы заблокировать доступ к странице или группе страниц, проще проверять их URL-адреса. Например, в определенные группы BuddyPress можно не пускать пользователей, которые не имеют подписки нужного уровня. Для этого в пользовательский плагин нужно добавить код следующего примера.

Пример

```
// Блокируем доступ к нашей группе
function my_buddy_press_members_group()
{
    $uri = $_SERVER['REQUEST_URI'];
    if(strtolower(substr($uri, 0, 16)) == "/groups/members/")
    {
        // убеждаемся в том в наличии подписки
        if(!pmpo_hasMembershipLevel())
        {
            wp_redirect(pmpo_url("levels"));
            exit;
        }
    }
}
```

```

    }
}
add_action("init", "my_buddy_press_members_group");

```

С момента выхода первого издания этой книги проект Paid Memberships Pro начал предлагать дополнение для интеграции с BuddyPress. Чтобы ограничить доступ к BuddyPress, используйте PMP Pro BuddyPress Add-on (oreil.ly/S00ok). Мы оставили приведенный пример, так как он хорошо иллюстрирует процесс блокирования любого общего URL-адреса в зависимости от наличия подписки.

Основную работу здесь выполняет функция `pmp_pro_hasMembershipLevel()`. Она может принимать два параметра. Первый — ID или название уровня подписки, который нужно проверить. Второй — ID пользователя, который запрашивает доступ. Если параметры не заданы, функция проверит, имеет ли текущий пользователь подписку *какого-либо* уровня.

Вы также можете выполнять отрицательные проверки, передавая, к примеру, `-1` в качестве ID уровня. Функция `pmp_pro_hasMembershipLevel(-1)` вернет `true`, если у текущего пользователя *нет* уровня 1. Если передать `0`, функция вернет `true`, если у текущего пользователя вообще нет подписки (вы также можете выполнить `!pmp_pro_hasMembershipLevel()`).

Вы можете передать ID и названия нескольких уровней внутри массива. Следующий фрагмент кода позволяет проверить наличие подписки уровня 1 или 2:

```

if (pmp_pro_hasMembershipLevel(array(1,2)))
{
    // какие-то действия для подписчиков уровня 1 или 2
}

```

- ◆ **Блокирование части страницы с помощью короткого кода.** Еще один способ ограничения доступа к контенту состоит в использовании коротких кодов в теле поста. В следующем примере показан фрагмент страницы, который выводит разные сообщения подписчикам разного уровня.

Пример

```

Welcome to SchoolPress!

[membership level="1"]Thanks for your continuing membership.[/membership]

[membership level="-1"]Sign up your school now![/membership]

```

Короткий код `[membership]` довольно прост. Как и функция `pmp_pro_hasMembershipLevel()`, он принимает параметр, который может быть ID или названием уровня. ID уровней могут быть нулевыми и отрицательными. Любой контент внутри короткого кода будет выводиться в зависимости от заданного уровня. Вы можете передать идентификаторы нескольких уровней, разделив их запятыми.

- ◆ **Блокирование части страницы в PHP-коде с помощью функции `pmp_pro_hasMembershipLevel()`.** Блокируя доступ к группе BuddyPress, мы использо-

вали функцию `pmpo_hasMembershipLevel()`. Эта же функция в шаблонах страниц и других участках приложения позволит ограничивать доступ к контенту или блокам кода. Допустим, в вашем заголовке может находиться код такого примера.

Пример

```
<?php if(is_user_logged_in()) { ?>
<div class="user-welcome">
    Welcome
    <?php if(function_exists("pmpo_hasMembershipLevel")
        && pmpo_hasMembershipLevel()) { ?>
        <a href="<?php echo pmpo_url("account"); ?>">
        <?php echo $current_user->display_name;?>
        </a>
    <?php } else { ?>
        <a href="<?php echo home_url("/wp-admin/profile.php"); ?>">
        <?php echo $current_user->display_name;?>
        </a>
    <?php } ?>
</div> <!-- end user-welcome -->
<?php } ?>
```

Этот код выводит подписчикам ссылку на их учетную запись в Paid Memberships Pro. Пользователям, у которых нет подписки, выводится ссылка на страницу их профиля в WordPress.

Шаг 8: изменение поведения Paid Memberships Pro

Далее мы опишем несколько распространенных способов изменения поведения Paid Memberships Pro. В целом, какой бы плагин мы ни взяли, эта процедура состоит из четырех шагов:

1. Определитесь с тем, что вы хотите изменить.
2. Выясните, где реализовано соответствующее поведение по умолчанию.
3. Найдите или добавьте хук для внесения нужных вам изменений.
4. Напишите действие или фильтр для использования этого хука.

Блокирование для пользователей без подписки любых страниц, кроме домашней

По умолчанию Paid Memberships Pro блокирует только те участки сайта, которые вы указываете. Но у некоторых сайтов должен быть ограниченный публичный доступ (например, чтобы посетители могли открывать только страницы с продажами, контактами и информацией о компании). Для этого посетителей без подписки, которые заходят на неразрешенные страницы, можно куда-то перенаправлять. Используйте код следующего примера.

```
function my_template_redirect()
{
    $okay_pages = array(
        pmpo_getOption('billing_page_id'),
        pmpo_getOption('account_page_id'),
        pmpo_getOption('levels_page_id'),
        pmpo_getOption('checkout_page_id'),
        pmpo_getOption('confirmation_page_id')
    );

    // если у пользователя нет подписки, отправляем его на домашнюю страницу
    if(!is_user_logged_in()
        && !is_home()
        && !is_page($okay_pages)
        && !strpos($_SERVER['REQUEST_URI'], "login"))
    {
        wp_redirect(home_url('wp-login.php?redirect_to=' .
            urlencode($_SERVER['REQUEST_URI'])));
    }
    elseif(is_page()
        && !is_home()
        && !is_page($okay_pages)
        && !pmpo_hasMembershipLevel())
    {
        wp_redirect(home_url());
    }
}

add_action('template_redirect', 'my_template_redirect');
```

В приведенном примере мы подготавливаем массив с ID постов, которые не должны быть доступны посетителям без подписки. Мы вызываем функцию `pmpo_getOption()`, чтобы получить ID страниц, сгенерированных плагином PMPPro, и открываем доступ к домашней странице с помощью встроенной в WordPress функции `is_home()`. Мы также делаем доступными любые страницы со словом `login` в URL-адресе; в нашей конфигурации это лишь страница входа в систему.

Блокирование доступа к файлам

К некоторым страницам, которые вы блокируете, могут быть прикреплены изображения или какие-то другие файлы. Если страница заблокирована, ваши пользователи их увидят. Но, если пользователь знает прямой путь к файлу, он сможет его загрузить, не являясь подписчиком. Дело в том, что при обработке URL-адресов

вроде `schoolpress.me/wp-content/uploads/logo.png` Apache выдает файлы пользователям напрямую, не сверяясь с PHP или WordPress.

Это поведение можно изменить, добавив в файл `*.htaccess` вашего сайта правило, которое пропускает любой URL-адрес вроде приведенного выше через специальный скрипт, входящий в состав Paid Memberships Pro. Откройте файл `*.htaccess` и добавьте перед остальными правилами переопределения следующий фрагмент кода:

```
RewriteEngine On
RewriteBase /
RewriteRule ^wp-content/uploads/(.*)$ \
    /wp-content/plugins/paid-memberships-pro/services/getfile.php [L]
```

Как это работает? WordPress позволяет прикреплять к посту изображения и файлы. Все эти файлы, которые в терминологии WordPress называются *вложениями*, хранятся в папке `/wp-content/uploads/` и связываются с постом, к которому они прикреплены, посредством записи в таблице `wp_posts`.

Записи таблицы `wp_posts`, относящиеся к вложениям, имеют значение `attachment` в столбце `post_status` и ID соответствующего поста в столбце `post_parent`.

Скрипт `getfile.php` находит в таблице `wp_posts` запись запрашиваемого файла и, если родительский пост вложения требует подписки, выдаст файл в том случае, когда к нему обращается пользователь с подходящим уровнем доступа.

Изменение пользовательских ролей в зависимости от уровня подписки

В большинстве примеров, представленных в этом разделе, предполагается, что подписчики имеют доступ только к клиентской части вашего приложения. Но иногда, если вы хотите назначить пользователю роль `"author"`, чтобы он мог вести блог, или какую-либо другую, отличную от `"subscriber"`, у него должна быть возможность работать с панелью управления WordPress.

В коде следующего примера роль `"author"` назначается любому новому подписчику определенного уровня. Если пользователь лишается соответствующего уровня подписки, его роль автоматически меняется на `"subscriber"`.

Пример

```
function my_pmpo_after_change_membership_level($level_id, $user_id)
{
    if($level_id == 1)
    {
        // Новый подписчик уровня #1.
        // Если у него есть роль subscriber, меняем ее на author.
        $wp_user_object = new WP_User($user_id);
        if(in_array('subscriber', $wp_user_object->roles))
            $wp_user_object->set_role('author');
    }
}
```

```

else
{
    // Это не подписчик уровня #1.
    // Если у него есть роль author, меняем ее на subscriber.
    $wp_user_object = new WP_User($user_id);
    if(in_array('author', $wp_user_object->roles))
        $wp_user_object->set_role('subscriber');
}
}
add_action(
    'pmpo_after_change_membership_level',
    'my_pmpo_after_change_membership_level',
    10,
    2
);

```

Подробнее о пользователях и ролях можно почитать в *главе 6*.

Международные и длинные адреса

По умолчанию адрес в форме оформления заказа Paid Memberships Pro состоит из таких отдельных полей, как "город", "штат" и "почтовый код". Внизу формы находится раскрывающийся список, в котором можно выбрать страну (рис. 15.9).

Рис. 15.9. Международный платежный адрес в Paid Memberships Pro

Возможно, вам нужно поменять страну по умолчанию, отредактировать список стран или сделать некоторые поля адреса необязательными. Далее приведен пример кода, который это делает.

Пример

```
/*
    Меняем страну по умолчанию с US (США) на GB (Великобритания)
*/
function my_pmpo_default_country($default)
{
    return 'GB';
}
add_filter('pmpo_default_country', 'my_pmpo_default_country');

/*
    Добавляем/удаляем страны в списке по умолчанию.
*/
function my_pmpo_countries($countries)
{
    // Убираем США
    unset($countries['US']);

    // Добавляем Луну (LN – сокращенно от Lunar?)
    $countries['LN'] = 'The Moon';

    // Вы также можете сформировать весь массив с нуля.
    // $countries = array('CA' => 'Canada', 'US' => 'United States',
    // 'GB' => 'United Kingdom');

    return $countries;
}
add_filter('pmpo_countries', 'my_pmpo_countries');

/*
    (необязательно) Для добавления/удаления стран
    в списке можно использовать фильтр pmpo_countries.
    Массив имеет следующий формат: array('US'=>'United States',
    'GB'=>'United Kingdom');, где ключом выступает короткий код
    страны, а значением – ее полное название.
*/

/*
    Делаем некоторые поля оплаты необязательными.
    Поля по умолчанию:
```

```
bfirstname, blastname, baddress1, bcity, bstate, bzipcode, bphone, bemail, bcountry,
CardType, AccountNumber, ExpirationMonth, ExpirationYear, CVV
*/
function my_pmpo_required_billing_fields($fields)
{
    // убираем штат и почтовый код
    unset($fields['bstate']);
    unset($fields['bzipcode']);

    return $fields;
}
add_filter('pmpo_required_billing_fields', 'my_pmpo_required_billing_fields');
```

Мобильные приложения на платформе WordPress

Мобильные приложения для iOS и Android, в которых используется WordPress? Почему бы и нет? У WordPress есть множество возможностей, которые можно по-разному применять как в нативных, так и в гибридных мобильных приложениях.

Но прежде чем погружаться в эту тему, давайте подумаем, для чего такие приложения могут применяться. Конечно, это необъятная область, но мы сосредоточимся на проектах, ориентированных на контент. Мы также обсудим некоторые аспекты разработки гибридных мобильных приложений, о которых вы должны знать.

Сценарии использования мобильных приложений

Итак, у нас есть мобильные приложения, способные выводить контент нашего сайта. Представим себе, насколько интерактивными они могут быть? Если использовать WordPress, степень интерактивности может быть любой на ваш выбор! Все, что можно делать с WordPress, доступно в гибридном мобильном приложении:

- ◆ Пользователи вашего приложения могут аутентифицироваться с помощью Facebook, Twitter или других социальных сетей с поддержкой технологии единого входа (англ. single sign-on или SSO).
- ◆ Если ваш сайт является интернет-магазином на основе WooCommerce, то ваши пользователи смогут просматривать и покупать товары прямо в мобильном приложении. Возможно, ваше приложение будет поддерживать разные интернет-магазины, размещенные в вашей сети; в таком случае продавцы смогут с его помощью управлять каталогом продукции и загружать фотографии товаров прямо со своих устройств в режиме реального времени.
- ◆ Возможно, вы создали веб-сайт знакомств на основе BuddyPress. Разработка фирменного мобильного приложения для этого сайта поможет вам увеличить число подписчиков. Благодаря таким встроенным функциям мобильных устройств, как камера, пользователи вашей социальной сети смогут делать селфи и загружать их прямо в свой профиль.
- ◆ Возможно, вы разработали сайт недвижимости и хотите, чтобы риелторы могли создавать посты типа "Homes" с описанием того, что они хотят продать. А поль-

зователи веб-приложения смогут публиковать фотографии с геолокацией, чтобы они были видны в Google Maps.

- ♦ Вы можете создать приложение для строительных подрядчиков, водопроводчиков, специалистов по ландшафтному дизайну, электриков, врачей, стоматологов, адвокатов, бухгалтеров или других людей, работающих в сфере услуг, чтобы помочь им отслеживать вакансии, просматривать информацию о клиентах, загружать фото и видео в стиле "до и после", проверять местонахождение заказчиков с помощью GPS, совместно работать со своими коллегами и многое другое.
- ♦ Возможно, вы уже вложили время и деньги в какую-то систему на основе WordPress и теперь хотите сделать ее доступной в разных магазинах мобильных приложений.

Перед вами открываются безграничные возможности, особенно если вы внедрите в свое гибридное мобильное приложение нативные функции. В случае необходимости всегда можно разработать полностью нативное приложение и сделать так, чтобы оно обращалось к сайту на основе WordPress; таким образом ваши контент и пользовательская база будут доступны на всех платформах.

Нативные и гибридные мобильные приложения

WordPress REST API — предпочтительный способ интеграции WordPress как с полностью нативными, так и с гибридными мобильными приложениями, который позволяет оптимизировать их производительность. Если вы опытный или начинающий разработчик нативных приложений, этот метод, скорее всего, будет для вас оптимальным. Веб-разработчикам, которые не собираются изучать Objective-C, Swift, Kotlin и/или Java, лучше остановиться на гибридных мобильных приложениях, и эта глава как раз для них.



В зависимости от потребностей конкретного проекта и ваших навыков (или навыков ваших коллег по команде разработки), гибридное мобильное приложение не всегда является лучшим выбором.

Что такое нативное мобильное приложение?

Нативным называют мобильное приложение для смартфонов и/или планшетов, которое написано на конкретном языке программирования для заданной операционной системы:

- ♦ Objective C = iOS;
- ♦ Java = Android.

Вот некоторые преимущества нативных мобильных приложений:

- ♦ Производительность — нативные приложения создаются и оптимизируются для определенных платформ с использованием системных языков, библиотек и API-интерфейсов, что делает их очень быстрыми и отзывчивыми.

- ◆ **Нативные библиотеки** — прямой доступ к нативным библиотекам дает огромное преимущество по сравнению с применением оберток вокруг стандартных возможностей.
- ◆ **Безопасность** — нативные приложения могут быть безопасней гибридных, так как они зависят только от платформы, в то время как последние задействуют различные технологии вроде JavaScript, HTML и CSS.

У подобных приложений также есть несколько недостатков:

- ◆ **Медленная разработка** — чтобы опубликовать приложение в более чем одном магазине, вам придется собирать один и тот же код по несколько раз.
- ◆ **Повышенные денежные расходы** — вдобавок ко времени, которое приходится тратить на сборку приложения под разные платформы, вам придется раскошелиться на нативных разработчиков под iOS и Android, у которых, как правило, более высокие зарплаты.
- ◆ **Ограниченный выбор программистов** — веб-разработчиков куда больше, чем тех, кто умеет создавать нативные мобильные приложения.

Что такое гибридное мобильное приложение?

Гибридные мобильные приложения, как и нативные, работают на смартфонах и планшетах, но они написаны на основе таких веб-технологий, как JavaScript, HTML и CSS. Гибридные приложения выполняются внутри нативного контейнера или обертки и используют браузерный движок устройства для отображения HTML и локальной интерпретации JavaScript. Для доступа к таким стандартным функциям, как акселерометр, камера, GPS и локальное хранилище, можно создавать и применять нативные плагины на JavaScript. Гибридное приложение, в сущности, представляет собой веб-сайт внутри нативной программы, который выглядит и ведет себя, как другие программы на этой платформе.

Почему стоит создавать гибридные приложения вместо нативных?

В наши дни более чем две трети мобильных разработчиков предпочитают создавать гибридные приложения вместо нативных. Растущая популярность этого выбора во многом объясняется экономией времени и денежных средств. Представьте, сколько времени придется потратить вашей компании на разработку, поддержку и обслуживание совершенно нового веб-сайта и нативных приложений под iOS и Android. Если реализовать похожий опыт взаимодействия с одинаковой функциональностью три раза, придется поддерживать три кодовые базы. Разработка гибридных приложений обычно занимает намного меньше времени, поскольку одна и та же кодовая база (скажем, WordPress) пригодна для реализации возможностей на разных платформах, таких как iOS и Android. Конечно, в некоторых случаях это выглядит как банальная загрузка веб-сайта внутри приложения, но если в итоге функциональность не страдает, и если ваш бизнес или бизнес ваших клиентов может быть пред-

ставлен таким образом в магазинах мобильных платформ, это будет хорошим экономичным решением.

Вот некоторые преимущества разработки гибридных мобильных приложений с использованием WordPress:

- ♦ Применение уже имеющихся навыков — вы и/или ваши коллеги уже заняты разработкой под WordPress. Вместо того чтобы учиться создавать нативные приложения, уделите внимание реализации новых крутых возможностей.
- ♦ Общая кодовая база. Зачем изобретать велосипед? Чтобы добавить нестандартную функциональность в свое мобильное приложение, зайдите в репозиторий плагинов WordPress или напишите собственный плагин, который будет делать то, что вам нужно. Пусть в основе вашего вебсайта и ваших приложений для iOS и Android лежит одна и та же кодовая база.
- ♦ Однородный опыт взаимодействия — веб-технологии, такие как JavaScript, HTML и CSS, позволят вам легко обновлять и контролировать опыт взаимодействия на всех платформах, на которых работает ваше приложение.
- ♦ Задел на будущее — постоянно развивающиеся веб-технологии и стеки разработки делают создание и обновление мобильных приложений все проще и проще. Это дает возможность легко охватывать существующие и будущие платформы.

Cordova

Apache Cordova — это открытый веб-фреймворк для создания гибридных мобильных приложений с использованием HTML, CSS и JavaScript.

PhoneGap

PhoneGap — это коммерческая версия Cordova, принадлежащая компании Adobe. Cordova и PhoneGap, по аналогии с WordPress.com и WordPress.org, имеют общую кодовую базу. При этом PhoneGap предоставляет простой в использовании компилятор и клиентскую поддержку.

Установка Cordova

Создайте директорию для своих проектов и перейдите в нее в своем терминале. Чтобы установить Cordova, вам понадобится Node.js (nodejs.org), открытая среда выполнения JavaScript. Введите в терминале команду `node -v`. Так вы сможете узнать текущую версию пакета Node.js и убедиться в том, что он установлен и готов к работе.

Для установки Cordova используется npm: выполните в терминале команду `sudo npm install cordova -g` (флаг `-g` нужен для глобальной установки). После того как пакет Cordova будет установлен, в терминале должна отобразиться информация о его текущей версии.

Чтобы создать новый проект, запустите в нужной вам директории команду `cordova create TestApp com.appresser.apps.testapp TestApp` с подходящими значениями. Команда `cordova create` принимает три параметра:

- ◆ название директории проекта — `TestApp`;
- ◆ имя пакета — `com.appresser.apps.testapp` (для этого принято использовать доменное имя, записанное в обратном порядке);
- ◆ название приложения — `TestApp`.

Давайте кратко пройдемся по файлам и директориям, которые создал фреймворк Cordova:

- ◆ `config.xml` — файл настроек, который по умолчанию задан для версий Cordova ниже 7;
- ◆ `package.json` — файл настроек, который по умолчанию установлен в Cordova, начиная с версии 7;
- ◆ `hooks` — хуки, представляющие собой пользовательские скрипты для изменения поведения команд Cordova;
- ◆ `platforms` — изначально эта директория пуста, но впоследствии в ней будут храниться код и файлы для интеграции Cordova с пакетами разработки под каждую платформу, такую как Android и iOS;
- ◆ `plugins` — плагины Cordova, которые необходимы вашему приложению; Здесь хранятся любые плагины, которые используют нативные возможности платформы;
- ◆ `www` — все ваши веб-ресурсы, которые будут завернуты в веб-представление.

Мы почти подошли к добавлению платформ в наш новый проект, но сначала следует свериться с соответствующими руководствами на веб-сайте Cordova и убедиться в том, что ваша среда удовлетворяет требованиям каждой платформы, для которой вы будете собирать свое приложение:

- ◆ Руководство по платформе Android (bwawwp.com/cordova-android-guide);
- ◆ Руководство по платформе iOS (bwawwp.com/cordova-ios-guide).



Полный список доступных консольных команд можно найти на странице bwawwp.com/cordova-cli.

Cordova и Android

Прежде чем добавить все необходимые файлы для работы с платформой Android SDK, убедитесь в том, что у вас есть доступ к пакету разработки под Android. Установите Android Studio (bwawwp.com/android-studio), если вы этого еще не сделали.



Если вы собираетесь использовать эмулятор Android, не забудьте установить его из Android SDK Manager (oreil.ly/50c_D).

После установки Android Studio выполните следующую команду в своем терминале, находясь в директории своего нового проекта:

```
cordova platform add android
```

В следующем примере приведен полученный результат.

Пример

```
Brians-MBP:TestApp bmess$ cordova platform add android
Using cordova-fetch for cordova-android@8.0.0
Adding android project...
Creating Cordova project for the Android platform:
  Path: platforms/android
  Package: com.appresser.apps.testapp
  Name: TestApp
  Activity: MainActivity
  Android target: android-28
Subproject Path: CordovaLib
Subproject Path: app
Android project created with cordova-android@8.1.0
Plugin 'cordova-plugin-whitelist' found in config.xml... Migrating it to
package.json
Discovered saved plugin "cordova-plugin-whitelist". Adding it to the project
Installing "cordova-plugin-whitelist" for android
Adding cordova-plugin-whitelist to package.json
```

Если заглянуть в директорию `platforms` вашего проекта, то можно заметить новую поддиректорию `android`, в которой содержатся все ресурсы, необходимые для работы приложения на платформе *Android*.

Если вы хотите скомпилировать свой новый проект, чтобы проверить его в работе, выполните в терминале команду `cordova build`. Эта команда соберет ваше приложение для всех доступных платформ. Вы также можете указать отдельную платформу, для которой нужно выполнить сборку, подав команду `cordova build android`.

Получив в терминале сообщение `BUILD SUCCESSFUL`, вы увидите путь к только что собранному файлу `*.apk`. Это ваше приложение, скомпилированное под Android. Вы можете установить и протестировать его на любом Android-устройстве. Если вы подключили свое устройство напрямую к компьютеру и сконфигурировали его для тестирования, то можете запустить на нем свое приложение, задав команду `cordova run android`.

Если у вас нет устройства с Android, то можно тестировать свое приложение в эмуляторе, выполнив команду `cordova emulate android`.

Cordova и iOS

По аналогии с тем, как Android Studio требует доступа к файлам Android SDK, для работы в Xcode вам понадобится пакет iOS SDK. Если вы еще не установили Xcode, сделайте это сейчас (иначе вы не сможете собрать iOS-приложение).

Чтобы добавить iOS в список поддерживаемых платформ вашего нового проекта, выполните в терминале команду `cordova platform add ios`. Вы должны получить ответ приведенный в следующем примере.

Пример

```
Brians-MBP:TestApp bmess$ cordova platform add ios
Using cordova-fetch for cordova-ios@5.0.0
Adding ios project...
Creating Cordova project for the iOS platform:
    Path: platforms/ios
    Package: com.appprasser.apps.testapp
    Name: TestApp
iOS project created with cordova-ios@5.0.1
Installing "cordova-plugin-whitelist" for ios
Brians-MBP:TestApp bmess$
```

Для сборки, запуска и эмуляции приложений под iOS предназначены те же консольные команды Cordova, что и под Android, только с `ios` в конце:

- ◆ `cordova build ios` — компилирует приложение, создавая готовую сборку;
- ◆ `cordova run ios` — запускает iOS-приложение в эмуляторе.

Плагины Cordova

Как вы уже знаете, у WordPress есть плагины для расширения функциональности вашего веб-сайта. Точно так же Cordova предлагает плагины для расширения возможностей вашего приложения. Эти плагины позволяют использовать стандартные функции устройства через JavaScript.

Для установки доступны как базовые, так и сторонние плагины, хотя вы можете создавать и свои собственные! В число основных плагинов Cordova входят следующие:

- ◆ `BatteryStatus` — предоставляет API-интерфейс для доступа к событиям, определяющим состояние аккумулятора, `batterystatus`, `batterycritical` и `batterylow`;
- ◆ `Camera` — API-интерфейс для создания фотоснимков и выбора изображений в галерее устройства;
- ◆ `Capture` — предоставляет доступ к возможностям устройства по записи/захвату звука, изображений и видео;
- ◆ `Connection` — выдает информацию о сотовом и Wi-Fi-соединениях устройства, позволяя узнать, подключено ли оно к Интернету;

- ◆ **Device** — предоставляет информацию об аппаратном и программном обеспечении устройства;
- ◆ **Events** — предоставляет различные прослушиватели событий, которые можно использовать в приложении, включая `deviceready`, `pause`, `resume`, `backbutton`, `menubutton`, `searchbutton`, `startcallbutton`, `endcallbutton`, `volumedownbutton`, `volumeupbutton` и `activated`;
- ◆ **File** — реализует File API, позволяя читать/записывать файлы, хранящиеся на устройстве;
- ◆ **Geolocation** — предоставляет информацию о местоположении устройства (широту и долготу);
- ◆ **Globalization** — выдает информацию о региональных настройках, языке и временной зоне устройства, позволяя выполнять соответствующие операции;
- ◆ **InAppBrowser** — дает возможность просматривать веб-страницы, не покидая приложение;
- ◆ **Media** — обеспечивает возможность записывать и проигрывать аудиофайлы;
- ◆ **Notification** — предоставляет доступ к некоторым нативным диалоговым элементам пользовательского интерфейса, таким как `alert`, `confirm`, `prompt` и `beep`;
- ◆ **Splashscreen** — отображает и скрывает экран-заставку во время запуска приложения;
- ◆ **Status Bar** — предоставляет функции для изменения панели состояния в iOS и Android;
- ◆ **Storage** — предоставляет функции для работы с локальным хранилищем устройства;
- ◆ **Vibration** — позволяет управлять виброрежимом устройства.

Со списком неосновных плагинов, доступных для Cordova, можно ознакомиться на странице bwawwp.com/cordova-plugins.

Чтобы добавить плагин Cordova в свой проект, выполните команду `cordova plugin add`, указав его название. Например, чтобы добавить плагин для работы с камерой, его полное название нужно указать в конце команды:

```
cordova plugin add cordova-plugin-camera.
```

Плагин также можно добавить по URL-адресу его Git-репозитория, например:

```
cordova plugin add https://github.com/apache/cordova-plugin-camera.git.
```

В любой момент, находясь в директории своего проекта, вы можете вывести список установленных плагинов, подав команду `cordova plugin ls`.

Ionic Framework

Ionic Framework — это открытый фреймворк для разработки мобильных приложений, который позволяет собирать нативные пакеты для iOS и Android, а также про-

грессивные веб-приложения, основанные на таких веб-технологиях, как JavaScript, HTML и CSS.

Ionic умеет работать с Cordova и содержит обширную библиотеку консольных команд, что позволяет вам выполнять практически любые команды Cordova, с которыми вы только что познакомились.

Последовательность работы.

1. Установите утилиту Ionic CLI, загрузив ее с сайта bwwawp.com/ionic-cli. Если у вас уже установлены Node.JS и npm, выполните команду:

```
sudo npm install -g ionic.
```

Проверьте установленную версию Ionic с помощью команды `ionic -v`.

Проверьте версию Node.js, выполнив `node -v`.

Для установки Cordova и Ionic подайте команду:

```
sudo npm install -g cordova ionic.
```

2. Установив Ionic, создайте новое приложение. Укажите его название, тип шаблона и фреймворк (здесь мы выбрали tabs, но существуют и другие типы компоновки), используя команду `start`:

```
ionic start SchoolApp tabs angular.
```

3. Зайдите в новую директорию проекта Ionic и просмотрите ее содержимое:

```
cd SchoolApp;  
ls.
```

4. Находясь в директории своего нового проекта, выполните команду `ionic serve`, чтобы собрать приложение и открыть его в веб-браузере. Вуаля! Ваше прогрессивное веб-приложение готово.

5. Чтобы использовать Ionic вместе с Cordova в своем проекте с поддержкой нативных возможностей, выполните команды:

```
ionic cordova platform add android;  
ionic cordova platform add ios.
```

Таким образом в ваш проект Ionic будут добавлены файлы Cordova для соответствующих платформ.

6. Выполните команду `ionic cordova emulate android`, чтобы запустить в эмуляторе свое новое приложение для Android.

То же самое можно сделать и в эмуляторе iOS:

```
ionic cordova emulate ios --no-native-run.
```

Если вы не указали эмулятор явно, вам, скорее всего, нужно будет запустить его заранее.

Теперь можно приступить к созданию нативных (iOS и Android) и прогрессивных веб-приложений с использованием WordPress! У вас есть несколько вариантов. Вы можете создать локальные экраны мобильного приложения, которые извлекают контент из WordPress REST API с помощью JavaScript; вы также можете загружать

страницы WordPress в веб-представлении, применяя специальную адаптивную тему оформления, предназначенную для мобильных устройств (ее можно купить или создать самостоятельно). У обоих методов есть свои преимущества; вы даже можете комбинировать их вместе в одном проекте, если это соответствует стоящим перед вами задачам.

Приложение-обертка

Возможно, вы уже реализовали в WordPress дополнительные функции (или планируете это сделать) и теперь хотите продублировать их в мобильном приложении. Чтобы убить двух зайцев одним выстрелом, проще и быстрее всего поместить существующий сайт на основе WordPress в *приложение-обертку*.

Обертка — это фактически нативное мобильное приложение с веб-представлением `iframe`. Иными словами, это просто веб-браузер, встроенный в само приложение. Этот браузер загружает URL-адрес вашего WordPress-приложения, которое в идеале должно использовать специальную адаптивную тему оформления, чтобы ваш сайт выглядел и работал, как обычное мобильное приложение. Вы можете добавить в свое гибридное приложение нативные функции, такие как доступ к камере, GPS, контактам и т. д., но, если вам требуется, чтобы его можно было загрузить в различных магазинах для мобильных устройств, и у вас нет времени, денег и ресурсов для разработки полноценного нативного или гибридного приложения на основе WordPress API, простой обертки будет достаточно. Почти все приемы с использованием платформы WordPress, ее плагинов и тем можно применять для мобильных приложений как в iOS, так и в Android. Представьте, какие возможности это открывает!

AppPresser

AppPresser (apppresser.com) — это один из самых простых способов создания мобильного приложения с использованием плагинов и тем WordPress. AppPresser App Builder позволяет визуальнo создавать мобильные приложения для iOS и Android без написания какого-либо кода. Хотя код, который AppPresser генерирует из App Builder, можно изменять и расширять по своему усмотрению.

AppPresser — это нечто большее, чем просто мобильная тема оформления с несколькими плагинами для WordPress; эта система основана на Cordova и Ionic Framework, поэтому она умеет интегрироваться с аппаратным обеспечением устройства, таким как камера, геолокация и акселерометр. Проекты, созданные с помощью AppPresser, можно загружать в iTunes и Google Play, как обычные нативные приложения.

Настройка проекта, основанного на AppPresser, не составляет труда. Эта система состоит из двух частей: вашего веб-сайта на основе WordPress и панели My AppPresser Dashboard.

Установка и конфигурация в WordPress

При регистрации на сайте AppPresser вы получаете доступ к теме оформления AppPresser Ionic, плагинам и панели управления My AppPresser. Первым делом

нужно установить и активировать основной плагин AppPresser. Войдите в свой веб-сайт на основе WordPress, перейдите в раздел **Plugins** → **Add New** (Плагины → Добавить новый) и поищите AppPresser. Установите и активируйте этот плагин, а затем загрузите тему AP3 Ion Theme на странице своей учетной записи в AppPresser. После этого проделайте то же самое для остальных плагинов, которые вы хотите использовать на своем веб-сайте.

Затем установите тему AppPresser AP3 WordPress в свою директорию с темами оформления. Перейдите в раздел **Appearance** → **Themes** → **Add New** → **Upload** (Оформление → Темы → Добавить новую → Загрузить) и загрузите соответствующий ZIP-файл. Эту тему не нужно активировать — она просто должна находиться в подходящей директории. Когда ваш вебсайт получит запрос от вашего мобильного приложения, он автоматически выдаст тему AppPresser вместо текущей.



Если при входе на сайт вы увидите, что тема AP3 Ion Theme является активной, это, вероятно, вызвано тем, что вы выполняли предпросмотр приложения и установили cookie. Это не ошибка, никто кроме вас не может видеть эту тему. Просто очистите свои cookie, и тема будет сброшена.

Чтобы сконфигурировать систему AppPresser и ее плагины, зайдите в меню настроек AppPresser на панели управления WordPress. Добавьте слаг вашего сайта и ID приложения, который можно получить, если войти на сайт **myapppresser.com** и перейти в раздел **Your App** → **General** (Ваше приложение → Общие). Добавьте и сохраните эти параметры.

У каждого плагина AppPresser, который вы устанавливаете и активируете, будет своя вкладка с конфигурационными параметрами. Некоторые плагины, в сущности, выступают связующим звеном между WordPress и плагинами Cordova для работы с камерой устройства, данными GPS и т.д. AppPresser поддерживает интеграцию с популярными сторонними плагинами для WordPress, такими как WooCommerce, LearnDash и BuddyPress, используя их REST API, чтобы генерировать страницы мобильного приложения с возможностью тонкой настройки. На сегодняшний день AppPresser предлагает такие плагины:

- ◆ AppBuddy;
- ◆ AppShare;
- ◆ AppCamera;
- ◆ AppWoo;
- ◆ AppCommerce;
- ◆ App Facebook Connect;
- ◆ AppCommunity;
- ◆ LearnDash;
- ◆ AppGeolocation;
- ◆ In App Purchases.
- ◆ AppPush;

App Builder

Подготовив плагины AppPresser на своем сайте, войдите в панель управления **myapppresser.com**. В разделе **Your app** (Ваше приложение) будут отображаться все ваши приложения. Нажмите оранжевую кнопку **New App** (Новое приложение), введите название и щелкните по кнопке **create app** (создать приложение).

После создания вашего приложения вы будете перенаправлены на его панель управления. Там вы сможете изменить его параметры, настроить уведомления и др. Нажмите кнопку **Customize and Build App** (Изменить и собрать приложение), чтобы изменить цвета своего приложения, добавить страницы, меню и прочее. Вам должен быть знаком этот процесс, так как это фактически редактор тем оформления для WordPress, только вместо предварительного просмотра активной темы выводится мобильное приложение (рис. 16.1).



Рис. 16.1. Изменение параметров приложения в MyAppPresser



MyAppPresser — это, в сущности, многосайтовая сеть WordPress, которая использует редактор тем оформления WordPress в качестве конструктора приложений.

В App Builder вы можете изменить свое мобильное приложение и собрать его в единое целое. Это позволяет вам почти полностью контролировать его внешний вид и поведение. Главное меню App Builder состоит из шести пунктов:

- ◆ **Colors** (Цвета) — визуальное изменение цветовой палитры любых элементов, доступных в вашем приложении. По умолчанию в число этих элементов входят Body Background, Text Color, Button Background, Button Text, Left Menu Background, Left Menu Text, Left Menu Icon Color, Link Color, Headings Color и Android Status Bar Background.
- ◆ **Design** (Дизайн) — изменение элементов дизайна вашего приложения, таких как шрифты заголовков, шрифт основного текста и любые дополнительные шрифты, которые вам необходимы. Здесь также можно задать любую пользовательскую таблицу стилей CSS, чтобы полностью изменить дизайн приложения.
- ◆ **Settings** (Настройки) — здесь можно указать общую информацию о приложении, которая понадобится при его сборке и дальнейшей настройке, и изменить его дополнительные параметры. Вы можете использовать боковое меню, вкладки или и то и другое. Вы можете открыть доступ к камере, push-уведомлениям и геолокации, загрузить значок приложения, экран-заставку, заголовочный логотип и любые другие статические ресурсы вроде локальных изображений или видеофайлов, которые нужно отображать или проигрывать. Вы можете загрузить файл JavaScript, чтобы выполнять собственный код внутри приложения. Вы можете указать язык, который будет использоваться по умолчанию и при необходимости изменить направление вывода текста.
- ◆ **Build and Preview** (Сборка и предварительный просмотр) — здесь вы можете собрать свое приложение после того, как закончите с изменением его параметров. Нажмите кнопку **Build App** (Собрать приложение) чтобы выполнить компиляцию для тестирования или загрузки в магазин приложений. Если вам нужно собрать проект для физического устройства, введите свой токен аутентификации PhoneGap Build. Это позволит устанавливать ваше приложение путем сканирования QR-кода. Стоит отметить, что для iOS требуется подписанный сертификат. Здесь также можно загружать и собирать файлы в виде ZIP-архивов, если вам нужно внести в приложение такие изменения, которые нельзя сделать с помощью App Builder.
- ◆ **Menus** (Меню) — здесь можно выбрать страницы, которые будут доступны в вашем приложении. Вы можете добавлять как пользовательские ссылки, так и пользовательские страницы. Укажите название и полный URL-адрес страницы своего веб-сайта на основе WordPress, которую вы хотите сделать доступной в своем приложении; таким образом любую страницу WordPress можно отобразить с использованием темы оформления AppPresser. Здесь также можно добавить любые дополнительные страницы.
- ◆ **Custom Pages** (Пользовательские страницы) — создание дополнительных страниц для приложения. Эти страницы будут загружаться прямо из вашего приложения, а не с сайта WordPress; они хранятся локально, и их можно по-разному настраивать.



Что касается пользовательских страниц в AppPresser, то это может быть как статический HTML-код, так и данные, загружаемые из API-интерфейсов. Пользовательские HTML-страницы не имеют прямого отношения к WordPress и могут содержать что угодно. Например, в приложение можно встроить страницу "О нас" с изображением и текстом, которая будет работать и без интернет-соединения. Вы также можете создать страницу со списком постов, который берется по REST API. При создании пользовательской страницы можно выбрать один из нескольких шаблонов, каждый из которых предоставляет различные поля для ввода такой информации как заголовок, URL-пути к REST API и/или текстовые поля с видеоизмененной разметкой в формате Ionic или HTML5.

Сочетание App Builder с темами и плагинами AppPresser позволяет быстро собирать приложения для iOS и Android из любого веб-сайта на основе WordPress. Представьте, что на вашем сайте установлен плагин подписок и LearnDash, и вы хотите интегрировать их и некоторые посты WordPress в свое приложение. Для начала установите основной и дополнительные плагины AppPresser, а также тему оформления. Это никак не отразится на вашем веб-сайте, поскольку AppPresser работает только в рамках приложения.

Дальше войдите в свой визуальный конструктор приложений и создайте новый проект. Выберите страницы своего сайта, которые вы хотите показывать в приложении. Например, если вам нужна лента активности BuddyPress, то можно добавить ее URL-адрес в меню приложения. Если вам нужны курсы LearnDash, можно создать для них пользовательскую страницу на основе REST API.

Вы можете добавить в свое меню кнопку входа, чтобы пользователи могли войти, выходить и регистрироваться на вашем сайте. После этого можно изменить цвета, добавить значок и экраны-заставки и собрать приложение. Вы можете посмотреть, как оно будет выглядеть, и протестировать его прямо в браузере, в приложении AppPresser Preview для iOS и Android или с помощью PhoneGap Build.

Компиляция и тестирование приложения

Во вкладке **Build and Preview** (Сборка и предварительный просмотр) на странице App Builder/theme customizer вам нужно будет настроить интеграцию с PhoneGap Build API с помощью токена аутентификации, сгенерированного в учетной записи PhoneGap Build. Это позволит отправить ваше приложение для дальнейшей компиляции и сборки. Если у вас нет учетной записи, ее можно бесплатно создать и затем перейти в раздел **Account** → **Client applications** (Учетная запись → Клиентские приложения), чтобы получить свой токен аутентификации. Вы должны сохранить этот токен в поле **PhoneGap Build Auth Token** (Токен аутентификации для PhoneGap Build) и нажать кнопку **Build App** (Собрать приложение). В результате все ваши программные файлы AppPresser будут упакованы и отправлены в сервис PhoneGap Build для дальнейшей компиляции. Как только PhoneGap закончит сборку, на экране появится QR-код и ссылка; это позволит вам загрузить и установить APK-файл для Android или iOS-приложение, если у вас есть подходящие ключ подписи и пароль. Если у вас еще нет учетной записи Apple-разработчика или ключа подписи для iOS, вы можете протестировать свой проект в iOS с помощью

AppPresser Preview (bwawwp.com/ap3preview-ios). Это приложение также можно использовать для тестирования проектов на Android-устройствах (bwawwp.com/ap3-preview-android).

AppPresser Preview App, в сущности, позволяет вам войти в AppPresser и протестировать его на своем устройстве в ходе сборки или реализации обновлений. Во вкладке **Build and Preview** (Сборка и предварительный просмотр) можно также заметить кнопку **Update Live App** (Обновить активное приложение). Нажмите ее, чтобы внедрить изменения (внесенные, к примеру, в пользовательские CSS или пункты меню с помощью App Builder) в существующее установленное приложение. Этим вы фактически просите приложение извлечь новые данные из своего API-интерфейса.

Представьте, к примеру, что у вас есть проект в магазине приложений и вы хотите поэкспериментировать, сделав цвет фона красным вместо синего. Но текущие пользователи вашего приложения не должны этого видеть. Внесите свои изменения в редакторе приложения и проверьте, как выглядит новый цвет. Если вы довольны, нажмите кнопку **Go Live** (Опубликовать), и фон вашего активного приложения станет красным. Любые локальные обновления вашего проекта, такие как новый статический контент (экраны-заставки, пользовательские HTML-страницы и т.д.), требуют перекомпиляции, дистрибуции и развертывания на устройствах, на которых установлено ваше приложение.



Чтобы увидеть какие-либо изменения, пользователи должны закрыть ваше приложение и запустить его заново.

Ссылки между страницами приложения

Существует несколько разных типов страниц, поэтому их связывание с помощью ссылок может принимать разные формы.

Чтобы страница WordPress ссылалась на страницу приложения, добавьте ссылку внутрь iframe-страницы WordPress, которая указывает на контент приложения. Для постов, страниц и CPT WordPress подойдет любой код, представленный далее.

◆ Ссылка на страницы в вашем боковом меню:

```
<a href="#" data-applink="2">Menu 2</a>
```

◆ Ссылка на вкладку:

```
<button data-apptablink="1">Tab Menu 1</button>
```

Число в ссылке обозначает номер пункта меню, начиная с нуля. Например, у первой страницы в меню будет номер 0, у второй — 1 и т. д. Ссылка ведет с одной пользовательской HTML-страницы на другую (которая должна быть в меню). В этом случае вы создаете пользовательскую HTML-страницу и ссылаетесь на контент приложения.

◆ Кнопка, ведущая на страницу:

```
<button ion-button>Visit Page</button>
```

◆ **Добавление функции щелчка мыши для этой кнопки:**

```
<button ion-button (click)="pushPage(pages.menus.items[0])">  
Visit Page  
</button>
```

Страница, на которую вы ссылаетесь, будет меняться в зависимости от вашего меню.

Для боковых меню укажите `pages.menus.items[]`, для вкладок — `pages.tab_menu.items[]`:

```
<button ion-button (click)="pushPage(pages.tab_menu.items[1])">  
Visit Page  
</button>
```

Число обозначает номер пункта меню, начиная с нуля. Первый пункт — 0, второй — 1 и т. д.

Если вам не нужна кнопка возврата, замените `pushPage` на `openPage`:

```
<button ion-button (click)="openPage(pages.menus.items[3])">  
Visit Page  
</button>
```

Плагин AppCamera

Расширение AppCamera позволяет делать фотоснимки, которые будут загружаться непосредственно на ваш сайт WordPress. Затем ваше приложение может их отображать. AppCamera также интегрируется с WooCommerce, BuddyPress и коротким кодом, который можно использовать в любом посте и на любой странице. После установки и активации этого плагина в настройках AppPresser появится новая вкладка **AppPresser Camera**. Откройте ее, чтобы настроить параметры расширения Camera.

Плагин AppCamera предлагает много разных параметров, с помощью которых можно изменить его поведение:

- ◆ **Camera license key** (Лицензионный ключ плагина) — укажите лицензионный ключ, который вы получили в результате покупки. Это позволит вам получать обновления плагина в момент их выхода.
- ◆ **Uploaded photos must be moderated** (Загруженные фотографии должны модерироваться) — задайте этот параметр, если вам нужна возможность проверять/одобрять/отклонять любые загружаемые фотографии перед их публикацией.
- ◆ **Email new photos to admin email** (Отправлять новые фотографии на почту администратора) — используйте этот параметр, если вы хотите получать на электронную почту администратора WordPress уведомления обо всех новых фотографиях, которые загружают пользователи, независимо от того, требуется модерация или нет.
- ◆ **Save photos to featured image** (Сохранять фотографии в список рекомендуемых изображений) — если включить этот параметр и сделать так, чтобы при загрузке

фотографий создавались новые посты, загружаемые изображения будут выводиться в списке рекомендуемых в соответствующем посте.

- ◆ **Photo upload description** (Описание загружаемых фотографий) — текст, который будет выводиться в качестве описания в динамической форме загрузки.
- ◆ **Text to display if logged out** (Текст, отображающийся при выходе) — текст, который будет выводиться пользователям, выходящим из системы.

После задания подходящих параметров нажмите кнопку **Save Settings** (Сохранить настройки).

AppCamera автоматически добавляет кнопку загрузки фотографий в фотогалерею товаров WooCommerce и ленту активности BuddyPress (для этого нужно, чтобы были установлены плагины AppBuddy и AppWoo).

Вы также можете вручную добавить шорткод (описанный далее) на страницу или в пост, который отображает в приложении кнопку для загрузки фотографий. Кнопки **Take Photo** (Сделать фотоснимок) и **Upload Image** (Загрузить изображение) выводятся только для аутентифицированных пользователей; остальные посетители видят текст **Please login** (Пожалуйста, войдите). Разрешать загрузку фотографий незарегистрированным пользователям небезопасно, поэтому в нашем плагине такая возможность исключена. Существует множество способов отображения загруженных фотографий в приложении или на веб-сайте (рис. 16.2).

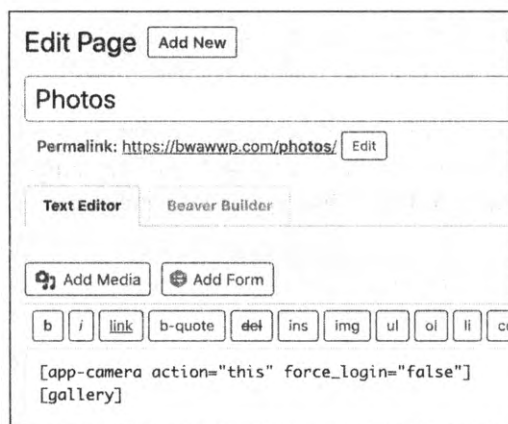


Рис. 16.2 Шорткод AppPhoto

На одной странице шорткодом `[app-camera]` можно разместить шорткод фотогалереи `[gallery]`, доступный в WordPress. Так можно отобразить ваши изображения. Обратите внимание на то, что эта страница не обновляется динамически при загрузке новых изображений, но при необходимости это можно реализовать с помощью JavaScript.

- ◆ **Action="this"** — если задать код `[app-camera action="this"]`, то каждая фотография, сделанная в приложении, будет загружаться в медиабibliothek и прикрепляться к текущему посту или странице, на которой размещен этот шорткод.

- ◆ Action="new" — если задать код [app-camera action="new"], то для каждой фотографии будет создаваться отдельный пост. Затем эти посты можно отобразить любым подходящим для вас способом: например, в виде пользовательского шаблона в вашей теме оформления или с помощью плагина, такого как Display Posts Shortcode.

Атрибут post_type позволяет указать тип поста, который вы уже зарегистрировали (такой как app_photos), чтобы выводить только эти фотографии, игнорируя остальные посты. Этот метод поддерживает модерацию.

Расширение AppCamera тоже предоставляет несколько шорткодов:

- ◆ [app-camera action="new"];
- ◆ [app-camera post_type="page"];
- ◆ [app-camera post_title="true"];
- ◆ [app-camera not_logged_in="Please log in first"];
- ◆ [app-camera description="You can upload your photos!"].

В одном шорткоде можно сочетать разные параметры, например:

```
[app-camera post_type="page" force_login="false" action="new"].
```

Вы можете сделать так, чтобы изображения модерировались перед публикацией на сайте. Для этого зайдите в настройки AppPresser, выберите вкладку **Camera** (Камера), установите флажок **Uploaded photos must be moderated?** (Модерировать загружаемые фотографии?) и сохраните установки. Если включить эту возможность, то соответствующий раздел появится только после добавления фотографий, которые подлежат модерации. Вы увидите раздел **Moderate Photos** (Модерировать фотографии) в меню AppPresser. Рядом с ним будет уведомление о том, сколько фотографий ожидают подтверждения. На эту страницу также можно зайти по ссылке **Photo Moderation Panel** (Панель модерации фотографий), которая появится рядом с кнопкой **Save Settings** (Сохранить настройки) во вкладке **AppCamera**.

Если взглянуть на страницу модерации (рис. 16.3), можно увидеть список ожидающих фотографий, доступных для просмотра, подтверждения и отклонения.

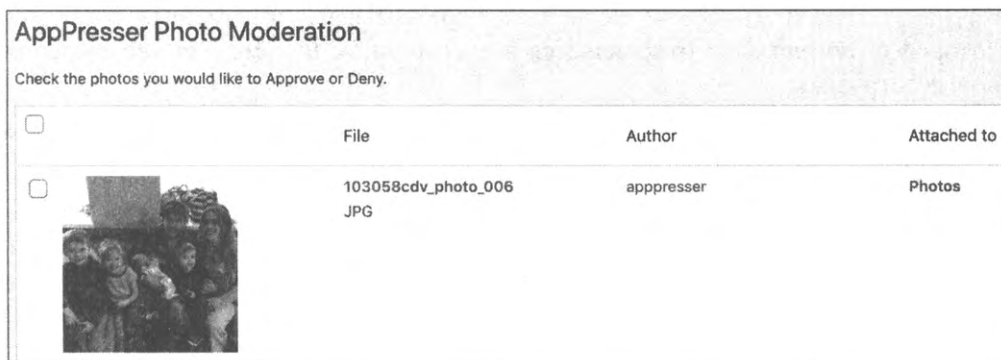


Рис. 16.3. Модерация загрузок

Каждая фотография содержит эскиз, имя/тип файла, имя автора, пост, в который она была загружена, и дату. Флажки слева позволяют подтверждать или отклонять сразу несколько элементов, но при этом у каждой фотографии есть отдельная ссылка подтверждения/отклонения. Как только список станет пустым, соответствующий пункт меню и ссылка исчезнут до тех пор, пока не появятся новые фотографии.

В состав AppCamera также входит множество хуков WordPress, с помощью которых можно изменить поведение приложения:

- ◆ `app_after_camera_buttons` — срабатывает непосредственно после отображения формы для загрузки фотографий;
- ◆ `app_after_process_uploads` — срабатывает непосредственно после завершения загрузки фотографии. Передает ID соответствующего поста и ID нового загруженного вложения. Это позволяет выполнять дополнительную обработку как вложения, так и поста;
- ◆ `app_before_camera_buttons` — срабатывает непосредственно перед отображением формы для загрузки фотографий;
- ◆ `active_plugins` — позволяет перехватывать функцию `get_option(active_plugins)` до того, как ваше расширение определит, активированы ли в настоящее время необходимые плагины. Значение по умолчанию: `get_option(active_plugins);`
- ◆ `app_camera_description` — позволяет перехватывать и при необходимости изменять текст, который выводится в качестве описания. Это значение изначально устанавливается в разделе **Camera** (Камера) на странице настроек AppPresser и равно `Upload your photos!`;
- ◆ `app_camera_not_logged_in_text` — дает возможность перехватывать и при необходимости изменять текст, который выводится незарегистрированным пользователям. Это значение изначально устанавливается в разделе **Camera** (Камера) на странице настроек AppPresser и равно `Upload your own customer image!`;
- ◆ `app_camera_post_title_label` — позволяет перехватывать и изменять метку поля `post_title`, которое выводится с помощью короткого кода `[app-camera]`. Значение по умолчанию: `<label> . __(Title:, app) . </label>;`
- ◆ `app_upload_email_message` — позволяет перехватывать и изменять сообщение, которое по умолчанию отправляется в электронном письме для уведомления о новых загрузках;
- ◆ `app_upload_email_subject` — обеспечивает возможность перехватывать и изменять тему писем, которая по умолчанию устанавливается для уведомлений о новых загрузках. Значение по умолчанию: `A new photo was uploaded;`
- ◆ `app_upload_email_to` — позволяет перехватывать и изменять адрес электронной почты, на который по умолчанию отправляются уведомления о новых загрузках. Значение по умолчанию: `get_set tings(admin_email);`
- ◆ `app_camera_photo_blog_embedded_img_size` — позволяет перехватывать и изменять размер по умолчанию для изображений блога. Значение по умолчанию:

`array(768, 2500, false)`. Первые два параметра — ширина и высота в пикселях, а третий определяет, нужно ли обрезать изображение;

- ♦ `appp_moderate_maybe_publish` — дает возможность перехватывать и при необходимости изменять массив аргументов, которые используются для публикации поста, когда фотография успешно проходит модерацию. Значение по умолчанию: `array(ID => absint($parent[0]), post_status => publish)`;
- ♦ `appp_insert_photo_post` — размещает загруженное изображение в область контента поста, если пользователь решил не добавлять его в список рекомендуемых.

Плагины WooCommerce

Плагин AppWoo был разработан специально для того, чтобы интегрировать в ваше приложение WooCommerce вместе с темой оформления AppPresser. Чтобы создать страницу WooCommerce в своем приложении, откройте меню изменения параметров. Нажмите кнопку **add items** (добавить элементы), выберите **WordPress/external links** (Ссылки на WordPress или внешние страницы), добавьте URL-адрес вашего магазина и сохраните установки. Вы также можете добавлять другие страницы, такие как учетная запись или пользовательская категория товаров.

Плагин AppWoo довольно автономный и не предлагает никаких пользовательских настроек. Однако внутри он позволяет оптимизировать AppTheme и WooCommerce под ваше нативное приложение:

- ♦ Изменяет структуру страницы отдельных товаров, делая ее похожей на настоящее приложение. Это достигается за счет удаления различных хуков обратного вызова, предоставляемых плагином WooCommerce, и регистрации некоторых новых хуков для AppTheme.
- ♦ Предоставляет интеграцию с AppSwiper и помогает работать с Ajax и ползунками для изображений товаров. Это упрощает отображение нескольких разных изображений товара для ваших клиентов.
- ♦ Автоматически дает возможность пользователям загружать и назначать изображения для каждого товара, если активирован плагин AppCamera.
- ♦ Выводит профиль пользователя в левой панели для зарегистрированных посетителей. При нажатии на меню выводится имя и аватар пользователя.
- ♦ Выводит клиентам информацию о содержимом корзины и общую сумму заказа в левой панели. Клиенты могут легко просматривать свои заказы прямо в меню.

AppWoo поддерживает несколько хуков, которые можно использовать из собственного пользовательского плагина для WordPress:

- ♦ `appp_after_product_images` — срабатывает непосредственно после вывода галереи изображений товара;
- ♦ `active_plugins` — позволяет перехватывать функцию `get_option(active_plugins)` до того, как будет определено, активирован ли в настоящее время плагин WooCommerce. Значение по умолчанию: `get_option(active_plugins)`;

- ◆ `appresser_woocom_gallery_ids` — дает возможность перехватывать и при необходимости изменять массив с идентификаторами изображений, которые выводятся в галерее. Значение по умолчанию: `$gallery_ids`;
- ◆ `appresser_disable_woo_styles` — позволяет перехватывать и предотвращать отключение стилей WooCommerce. Значение по умолчанию: `true` //Styles will be disabled (стили отключаются).

Если вы применяете в своем приложении WooCommerce, то на странице товара будет автоматически отображаться значок корзины. Если вы хотите, чтобы он выводился на каждой странице, можете добавить код следующего примера в пользовательский файл JavaScript в плагине или дочерней теме на своем сайте WordPress.

Пример

```
jQuery(document).ready(function($) {
    if($('body').hasClass('woocommerce-page')) {
        var message_array = {};
        message_array['post_title'] = window.appp.post_title;
        message_array['cart_link'] = window.apppwoo.cart_url;
        parent.postMessage(JSON.stringify(message_array), '*');
    }
});
```

Имейте в виду, что это всего лишь пример. Чтобы этот скрипт как следует работал на вашем сайте, вам, возможно, придется его отредактировать.

В последние несколько лет для WordPress и WooCommerce были выпущены огромные пакеты обновлений, поэтому разработчики AppPresser создали новый плагин для интеграции WooCommerce в мобильные приложения. Он называется *AppCommerce* и отличается повышенной производительностью, изящностью и акцентом на обеспечение как можно лучшего опыта взаимодействия с вашим магазином. AppCommerce подключается к магазину через WooCommerce REST API. Ваши клиенты могут добавлять товары в корзину, аутентифицироваться, просматривать прошлые заказы и учетную информацию, создавать списки желаний и т. д. Оплата заказа происходит на вашем веб-сайте, который открывается в браузере приложения. Это делается прозрачно, благодаря чему вы можете использовать свои текущие параметры платежных систем, доставки и налоговых отчислений без дополнительной конфигурации. При желании оформление заказа можно полностью изменить, чтобы оно выглядело не так, как на веб-сайте. После оплаты ваши клиенты автоматически перенаправляются обратно в приложение. Поведение AppCommerce можно менять как угодно с помощью конструктора приложений AppPresser. Вы выбираете страницы, которые должны быть в приложении, какие товары будут отображаться, какие цвета выбрать и многое другое.

Чтобы добавить магазин, корзину и страницы профиля WooCommerce в свое приложение, откройте настройки его параметров и перейдите в раздел **Custom Pages**

(Пользовательские страницы). Там вам нужно нажать кнопку **Add New Page** (Добавить новую страницу) и выбрать **AppCommerce**.

Чтобы добавить список товаров на страницу магазина или любую другую страницу приложения, достаточно воспользоваться тегом `<woo-list>` и выбрать среди компонентов **Shop** (Магазин). Следующий код даст вам возможность вывести на своей пользовательской странице список товаров:

```
<woo-list route="products" infiniteScroll="true"></woo-list>
```

Чтобы изменить отображаемые товары или их категории, используйте URL-параметры, которые можно найти в документации WooCommerce REST API (bwwwp.com/woocom-api). Например, чтобы выводить только простые товары, можете добавить условие `"products?type=simple"`, а с помощью `"products?stock_status=outof stock"` можно создать список товаров, которых нет в наличии:

```
<woo-list route="products?stock_status=outofstock" infiniteScroll="true">
</woo-list>
```



Если вы использовали шаблон приложения AppCommerce, то у вас уже могут быть эти страницы. В этом случае их не нужно создавать заново.

Чтобы отдельно вывести в приложении корзину, на одной из страниц следует разместить тег `<woo-cart>` и выбрать в качестве компонента **Cart** (Корзина):

```
<woo-cart></woo-cart>
```

Для страницы учетной записи используйте следующее:

```
<woo-account></woo-account>
```



Помните, что магазин, корзина и учетная запись являются обязательными страницами приложения.

Вы можете добавить в свое приложение другие страницы WooCommerce, такие как список желаний. Клиенты могут сохранять в него товары, щелкая по ссылке **Add to list** (Добавить в список). Создайте новую пользовательскую страницу и выберите AppCommerce. Укажите в качестве компонента **None** и добавьте следующий код:

```
<woo-list wishlist="true"></woo-list>
```

Оформление заказа происходит на вашем веб-сайте, открытом в браузере, встроенном в приложение. Для обеспечения наилучшего опыта взаимодействия мы советуем использовать плагин WooCommerce Smart Checkout.



Почти 40% всех покупок в Интернете во время праздничного сезона 2018 года было сделано на смартфонах.

LearnDash/AppLMS

Интеграция LearnDash позволяет добавлять в приложение курсы, темы, уроки и тесты. Вы можете ограничивать доступ к контенту для незарегистрированных посетителей, а также внедрять плагины подписки, BuddyPress, игрофикацию и т. д. Чтобы добавить LearnDash в приложение, достаточно создать страницу, которая отображает ваши курсы. Все остальное будет сделано за вас. Создайте в настройках параметров приложения новую пользовательскую страницу, на которой будет находиться список ваших курсов. Для этого выберите пункт **Custom pages** → **Add new page** (Пользовательские страницы → Добавить новую страницу) и вставьте следующий код:

```
<ap-list wp="true" card="true" class="col-2"
  route="https://mysite.com/wp-json/wp/v2/sfwd-courses?per_page=10">
  </ap-list>
```

Измените параметр `per_page`, чтобы выводить столько курсов, сколько вам нужно. Курсы будут отображаться в виде карт в два столбца. Вы также можете модифицировать компонент `ap-list`, чтобы уменьшить количество столбцов до одного:

```
<ap-list wp="true" route="https://mysite.com/wp-json/wp/
  v2/sfwd-courses" infiniteScroll="true"></ap-list>
```

Добавьте любой текст или HTML-код, который вам нужен, и сохраните. Затем добавьте эту страницу в свое меню и пересоберите приложение (рис. 16.4).

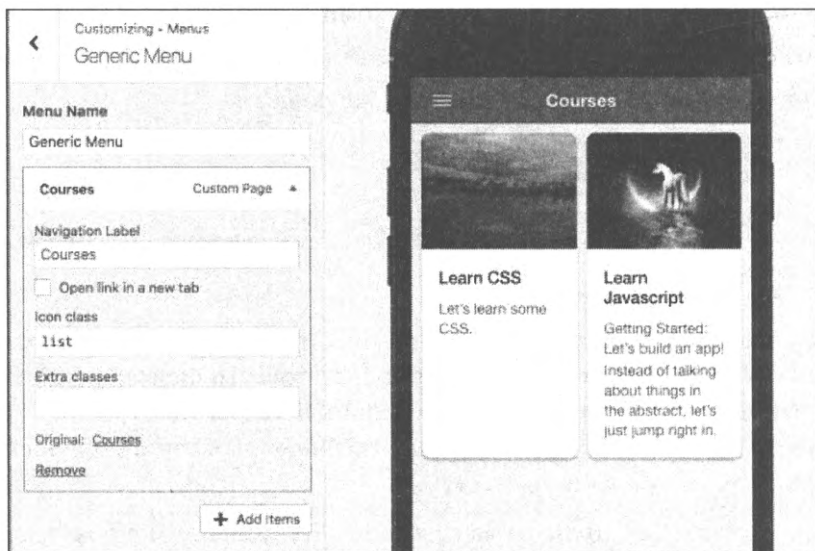


Рис. 16.4. Добавление курсов в приложение

Если вам не по душе этот подход на основе API-интерфейса, или если вам нужна большая гибкость, можно сделать то же самое с помощью WordPress. Создайте новую страницу на своем сайте WordPress и разместите на ней шорткод со списком курсов LearnDash. Если вы разработчик или просто хотите существенно изменить

поведение своего приложения, можете написать собственный плагин, который будет расширять имеющиеся возможности так, как вам нужно.

AppPush

Push-уведомления — это сообщения, доставляемые на каждое устройство, на котором установлено ваше приложение (при условии, что они включены в настройках). Для их рассылки требуются определенные приготовления, поэтому внимательно следуйте приведенным ниже инструкциям. Вот что следует отметить:

- ◆ Push-уведомления нельзя тестировать в симуляторе или в приложении AppPresser Preview; это можно делать только на настоящем устройстве.
- ◆ Большинство проблем связано с сертификатами iOS, поэтому строго следуйте инструкциям.
- ◆ У вас должна быть действительная учетная запись iOS-разработчика на сайте Apple. Подробнее об этом можно почитать по адресу **developer.apple.com**.
- ◆ Android требует наличие бесплатного проекта Firebase и API-ключа.

Подготовив сертификаты для iOS и проект Firebase, вы можете указать соответствующую информацию на сайте **myapppresser.com** и в PhoneGap Build, чтобы иметь возможность отправлять push-уведомления либо из раздела My AppPresser, либо с помощью плагина AppPush, установленного и интегрированного в ваш веб-сайт WordPress.



Если вы хотите узнать больше о проекте AppPresser и его подключаемых плагинах, перечисленных ранее, то можете ознакомиться со страницей **bwawwp.com/apppresser**.

PHP-библиотеки, интеграция веб-сервисов и миграция с других платформ

Время от времени у вас может возникать необходимость в интеграции возможностей, функций и данных, которые не имеют прямого отношения к WordPress.

В этой главе мы рассмотрим некоторые популярные PHP-библиотеки, веб-сервисы и API-интерфейсы, которые можно легко интегрировать в WordPress. Мы также обсудим структурированный процесс, с помощью которого в WordPress можно перенести контент/данные из практически любого стороннего ПО (это обычно называют *миграцией между платформами*). Перенесите в WordPress все, что только можно!

PHP-библиотеки

Большинство языков программирования, включая PHP, предоставляют модульные коллекции кода, классов и функций. Эти коллекции обычно называют *библиотеками* или *расширениями*.

Не изобретайте велосипед! Для выполнения разных узкоспециализированных задач существуют PHP-библиотеки, с помощью которых можно расширять создаваемые вами приложения.

Большая часть WordPress написана на PHP. Это означает, что, как WordPress-разработчик, вы можете использовать готовые или даже писать собственные PHP-библиотеки. Упорядоченный список некоторых популярных PHP-библиотек можно найти на странице bwwwp.com/phpfunct-ref.

В репозитории WordPress есть множество плагинов, которые взаимодействуют с различными библиотеками и внешними сервисами, поэтому, если вы хотите расширить возможности WordPress, сначала стоит заглянуть в этот репозиторий. Конечно, если вы ищите плагин, который делает именно то, что вы от него хотите, вам, возможно, придется написать его самостоятельно или на основе существующего кода.

Мы уже неоднократно упоминали GitHub. Это отличное место для поиска нужной вам функциональности. Еще одним хорошим источником готового кода могут послужить открытые проекты, написанные на PHP, такие как Drupal, Laravel, Joomla

и/или Magento. Возможно, то, что вы ищете, существует в виде модуля для Drupal, в таком случае возьмите этот код и интегрируйте его в WordPress.

Давайте рассмотрим несколько весьма полезных PHP-библиотек с расширенными возможностями, которые можно легко внедрить в WordPress.



Окружения некоторых веб-хостингов несовместимы с определенными PHP-библиотеками. Бывают библиотеки, которые служат обертками вокруг какого-то ПО, которое должно быть установлено на вашем веб-сервере. Если у вас нет доступа для конфигурации своего серверного окружения, вы не сможете их использовать.

Генерация и модификация изображений

Представьте себе самописный Photoshop, который можно использовать для создания таких вещей, как генераторы бессмысленных мемов (quickmeme.com/) или чего-то более продуктивного. Генерация и модификация изображений может пригодиться для целого ряда задач:

- ♦ изменение размеров и масштабирование изображений;
- ♦ сравнение изображений;
- ♦ копирование изображений;
- ♦ преобразование изображений в различные графические форматы;
- ♦ изменение цветов изображения;
- ♦ преобразование текста в графический вид и наложение его на изображения;
- ♦ объединение нескольких изображений;
- ♦ создание анимированных изображений;
- ♦ применение пользовательских графических фильтров (прямо как в Instagram);
- ♦ размытие и повышение резкости изображений;
- ♦ добавление рамок в изображения;
- ♦ динамическое построение диаграмм графиков на основе данных;
- ♦ отрисовка трехмерных изображений.

Как можно себе представить, у этих возможностей существует множество применений, учитывая, что программная модификация изображений почти ничем не ограничена. Выбор графической PHP-библиотеки зависит от того, для чего она вам требуется.

По умолчанию WordPress использует для работы с изображениями и в своей медиатеке одну из двух наиболее популярных PHP-библиотек, предназначенных для этих целей. Первым делом WordPress проверяет, доступна ли библиотека Imagick, и, если ее нет, ищет библиотеку GD. Если обе отсутствуют, то при попытках вызова функций WordPress, связанных с редактированием изображений, будет выводиться ошибка. WordPress задействует одну из этих библиотек в фоновом режиме при обрезании, повороте и любых других операций с графическими файлами.

Вы можете легко узнать, что применяется в установленной вами версии WordPress, Imagick или GD, выполнив следующий пример кода программы из файла `functions.php` своей темы оформления.

Пример

```
<?php
// Не делайте этого на реальном сайте, который вам важен.
function bwawwp_image_library_check(){
    if(extension_loaded('imagick')) {
        $imagick = new Imagick();
        print_r($imagick->queryFormats());
    } else {
        echo 'No ImageMagick!';
    }

    if(extension_loaded('gd')) {
        print_r(gd_info());
    } else {
        echo 'No GD!';
    }
    exit();
}
add_action("init", "bwawwp_image_library_check");
```

Узнав, какую библиотеку использует ваш сайт, вы можете легко приступить к написанию совместимого с ней кода.

GD

GD (bwawwp.com/gd) — это, наверное, самая популярная PHP-библиотека для работы с изображениями, которая обычно входит в стандартную поставку PHP. При редактировании больших изображений она, как правило, функционирует чуть быстрее, чем Imagick, и в целом занимает меньше серверных ресурсов. Это отличный инструмент, который способен удовлетворить большинство ваших нужд, однако Imagick отличается более широкими возможностями.

Imagick

Imagick (bwawwp.com/imagemagick) — это один из мощнейших инструментов для работы с изображениями, который входит в число наших любимых, хотя было бы здорово, если бы кто-нибудь перенес его веб-сайт WordPress. Imagick обеспечивает программное редактирование графических файлов практически любым образом, как вы только можете себе представить. Вы можете установить эту библиотеку на свой сервер и запускать ее из командной строки с помощью PHP-функций `shell_exec()` или `exec()`. Можно загрузить обертку для нее на языке PHP

(bwawwp.com/imagick-php). Она не поставляется вместе с PHP по умолчанию, и ее нужно устанавливать отдельно, вместе с самим пакетом Imagick.

Джастин Стернберг разработал на основе Imagick очень простые в использовании методы наложения текста на любое изображение с последующим сохранением результата в виде файла, который прикрепляется к посту WordPress. Они могут быть крайне полезными, если вам нужно, чтобы все изображения в вашем приложении содержали соответствующие URL-адреса; таким образом, если кто-то возьмет их без разрешения, все будут знать, где искать оригинал. Код Джастина можно посмотреть по адресу bwawwp.com/image-waterm.

Zebra_Image

Zebra_Image (bwawwp.com/zebraimage) — это чрезвычайно легковесная и простая в использовании PHP-библиотека для работы с изображениями. Это отличная альтернатива в случаях, когда вам нужно что-то попроще пакета Imagick (например, вы хотите выполнять лишь базовые операции, такие как изменение размера, обрезка, поворот, отражение и/или повышение четкости), или вы просто не можете установить Imagick на свой веб-сервер. Эта библиотека состоит из одного файла и требует наличия лишь расширения GD, которое обычно поставляется в скомпилированном виде вместе с PHP.

Imagine

Imagine (bwawwp.com/imagine) находится где-то между Imagick и Zebra_Image. Это очень мощная объектно-ориентированная PHP-библиотека, и она подойдет для любого рода операций с изображениями. Как и Zebra_Image, она основана на GD.

Dynamic Dummy Image Generator

Dynamic Dummy Image Generator (bwawwp.com/dummyimage) — это не PHP-библиотека, а прекрасный, хотя и небольшой инструмент, который мы хотели упомянуть. Он может помочь вам в разработке вашего веб-сайта на основе WordPress. Вам, наверное, знакома ситуация, когда клиент обещает прислать изображения, но, спустя три недели, у вас их все еще нет. Dummy Image (oreil.ly/T5Rb-) позволяет динамически создать на основе URL-адреса изображение любого размера.

Snappy

А вот это по-настоящему круто! С помощью Snappy (bwawwp.com/snappy) можно автоматически создавать изображения и/или PDF-файлы из любых URL-адресов или HTML-страниц. Только представьте себе, сколько всего интересного можно сделать, программно генерируя снимки любых веб-сайтов.

Генерация PDF

Программная генерация и редактирование PDF-документов может быть крайне полезной. Например, вам, наверное, приходилось заходить на веб-сайт школы или

мэрии с кучей файлов в формате PDF (таких как обеденные меню, календари, списки предстоящих событий, формы регистрации для участия в мероприятиях, списки рекомендуемой литературы, повестки/протоколы заседаний, справочники, отчеты, практические руководства и т. д.), которые, наверное, стоило бы оформить в виде обычных веб-страниц. Одна из ключевых причин, по которым школы и муниципалитеты используют PDF, состоит в том, что этот формат легко распространять через почтовые рассылки или в виде печатных копий. Конечно, иногда бывает полезно распечатать документ, но хранение контента исключительно в формате PDF имеет свои недостатки:

- ◆ PDF не соответствует закону о гражданах с инвалидностью, принятому в США. Если организация не делает свой контент доступным для всех, она рискует получить судебный иск.
- ◆ PDF не подходит для мобильных устройств. Документы этого формата можно просматривать на мобильных платформах, но это сродни просмотру неоптимизированного веб-сайта, на котором нельзя ничего прочитать без масштабирования.
- ◆ Если вы храните свой контент как на веб-странице, так и в PDF-файле, каждое его обновление придется дублировать.

Но что, если у каждой веб-страницы вашего сайта должна быть копия в формате PDF, доступная для загрузки? Или у вас есть большое количество онлайн-статистики и отчетов, из которых нужно генерировать PDF-документы? Как быть, если у вас постоянно обновляются события календаря и каждое изменение вам необходимо вносить в PDF-файл? Эти и аналогичные проблемы можно решить путем программной автоматизации процессов, которая сэкономит вам уйму времени и денег. Давайте поговорим о некоторых PHP-библиотеках, которые мы предпочитаем использовать для автоматической генерации PDF.

Snappy

Мы уже упоминали Snappy (bwawwp.com/snappy), но следует подчеркнуть, насколько легко и просто с помощью этого инструмента можно сгенерировать PDF-файл из веб-страницы по заданному URL-адресу или HTML-коду. Вы даже можете формировать многостраничные PDF-документы, указывая сразу несколько URL-адресов. Хотите получить подробный PDF-документ со всем содержимым вашего веб-сайта? Просто составьте список всех ваших страниц и передайте его Snappy!

Snappy использует две популярных утилиты командной строки для преобразования HTML в PDF и изображения: *wkhtmltopdf* и *wkhtmltoimage*. Вы можете легко загрузить и установить необходимые двоичные файлы на своем сервере, используя Composer — диспетчер зависимостей для PHP (getcomposer.org/).

FPDF

Буква F в аббревиатуре FPDF (fpdf.org) означает "free" (бесплатный)! Это отличная альтернатива проекту PDFlib, если вам не нужно создавать слишком уж много PDF-документов и если эти документы не слишком сложные. FPDF позволяет формиро-

вать динамические PDF-файлы с использованием чистого PHP: он не зависит ни от каких других PHP-библиотек, если не считать Zlib (для сжатия) и GD (для поддержки GIF), которые обычно вкомпилированы в PHP. Среди основных функций FPDF можно выделить следующие:

- ◆ `__construct` — конструктор;
- ◆ `AcceptPageBreak` — принимать или не принимать автоматические разрывы страниц;
- ◆ `AddFont` — добавить новый шрифт;
- ◆ `AddLink` — создать внутреннюю ссылку;
- ◆ `AddPage` — добавить новую страницу;
- ◆ `AliasNbPages` — создать псевдоним для определенного количества страниц;
- ◆ `Cell` — напечатать ячейку;
- ◆ `Close` — завершить документ;
- ◆ `Error` — неустраняемая ошибка;
- ◆ `Footer` — указать нижний колонтитул страницы;
- ◆ `GetPageHeight` — получить текущую высоту страницы;
- ◆ `GetPageWidth` — получить текущую ширину страницы;
- ◆ `GetStringWidth` — вычислить длину строки;
- ◆ `GetX` — получить текущую позицию по горизонтали;
- ◆ `GetY` — получить текущую позицию по вертикали;
- ◆ `Header` — получить заголовок страницы;
- ◆ `Image` — вывести изображение;
- ◆ `Line` — нарисовать линию;
- ◆ `Link` — вставить ссылку;
- ◆ `Ln` — задать перенос строки;
- ◆ `MultiCell` — вывести текст с переносами строк;
- ◆ `Output` — сохранить или отправить документ;
- ◆ `PageNo` — указать номер страницы;
- ◆ `Rect` — нарисовать прямоугольник;
- ◆ `SetAuthor` — указать автора документа;
- ◆ `SetAutoPageBreak` — указать режим автоматического разрыва страниц;
- ◆ `SetCompression` — включить или выключить сжатие;
- ◆ `SetCreator` — указать создателя документа;
- ◆ `SetDisplayMode` — указать режим отображения;
- ◆ `SetDrawColor` — указать цвет контура;

- ◆ SetFillColor — указать цвет заливки;
- ◆ SetFont — указать шрифт;
- ◆ SetFontSize — указать размер шрифта;
- ◆ SetKeywords — назначить документу ключевые слова;
- ◆ SetLeftMargin — указать отступ слева;
- ◆ SetLineWidth — указать толщину линий;
- ◆ SetLink — указать адрес для внутренней ссылки;
- ◆ SetMargins — указать отступы;
- ◆ SetRightMargin — указать отступ справа;
- ◆ SetSubject — указать тему документа;
- ◆ SetTextColor — указать цвет текста;
- ◆ SetTitle — указать название документа;
- ◆ SetTopMargin — указать отступ сверху;
- ◆ SetX — указать текущую позицию по горизонтали;
- ◆ SetXY — указать текущие координаты;
- ◆ SetY — указать текущую позицию по вертикали;
- ◆ Text — вывести строку;
- ◆ Write — вывести текст в виде потока.

Другие PHP-библиотеки для генерации PDF

В зависимости от ваших конкретных нужд вам, вероятно, стоит поискать в Интернете некоторые из следующих проектов и ознакомиться с ними более детально:

- ◆ wkhtmltopdf
- ◆ mPDF
- ◆ Dompdf
- ◆ TCPDF
- ◆ HTML2Pdf

Геолокация и геотаргетинг

Существует множество сервисов геолокации с разными возможностями, такими как определение приблизительного местоположения по заданному IP-адресу или приблизительной широты/долготы на основе предоставленных GPS-данных или сети Wi-Fi. Если у вас есть доступ к местоположению ваших посетителей, то становятся можно выполнять различные полезные действия:

- ◆ Ограничивать доступ к веб-сайту или отдельным страницам в зависимости от местоположения.

- ◆ Выдавать контент веб-сайта в зависимости от местоположения (например, рекламе с поддержкой геотаргетинга).
- ◆ Отслеживать местоположение человека или предмета в реальном времени.
- ◆ Находить местоположение ближайшего объекта на основе предоставленных координат.

Представьте, что авторам школьного веб-приложения нужен дополнительный уровень безопасности: доступ к странице входа в систему должны иметь только те, кто находится в одном городе или штате со школой. Такой тип защиты может потребоваться только в случае, если вы хотите ограничить аудиторию своего приложения определенной географической областью (или областями), но с его помощью можно существенно уменьшить количество потенциальных попыток взлома. Вы также можете заблокировать определенные регионы, такие как Китай. Возможно, вам заранее известно, что у посетителей из Китая не должно быть доступа к вашему приложению.

Возможно, веб-сайт вашего штата должен выводить при первой загрузке ближайшие к вам школы. Представьте, что каждая "школа" — это пост СРТ, в метаданных которого хранится соответствующий адрес. Того же результата можно добиться, если получить местоположение конечного пользователя и выполнить метазапрос для поиска школ, находящихся поблизости.

MaxMind GeoIP

MaxMind GeoIP (bwwawp.com/geo-ip) извлекает данные из IP-адреса пользователя, такие как его местоположение и интернет-провайдер. Это отличный сервис с развитым API-интерфейсом и базой данных, доступной для свободной загрузки.

Вы можете установить PHP-библиотеку, которая открывает доступ ко всем возможностям веб-сайта MaxMind. Чтобы использовать ее API-интерфейс, загрузите и установите код из GitHub (bwwawp.com/maxmindgeoip).

Geocoder PHP

Geocoder PHP (bwwawp.com/geocoder-php) — это PHP-библиотека, которая поможет вам создавать веб-приложения и мобильные приложения с поддержкой геолокации. В отличие от проекта MaxMind, который сам является провайдером геоинформации, Geocoder PHP позволяет вам выбрать один из нескольких провайдеров, включая MaxMind. Эта PHP-библиотека делает процесс запроса и извлечения геолокационных данных из разных источников удобным и однородным. Перед ее использованием нужно выбрать провайдера и HTTP-клиент/адаптер. Вот список доступных на сегодняшний день провайдеров (хотя при необходимости его можно расширить):

- | | |
|------------------|-----------------|
| ◆ Algolia Places | ◆ FreeGeoIp |
| ◆ ArcGIS Online | ◆ Geocode Earth |
| ◆ Bing Maps | ◆ GeoIP |

- | | |
|----------------------|------------------|
| ◆ GeoIP2 | ◆ Mapbox |
| ◆ Geonames | ◆ MapQuest |
| ◆ GeoPlugin | ◆ MaxMind |
| ◆ Google Maps | ◆ MaxMind Binary |
| ◆ Google Maps Places | ◆ Nominatim |
| ◆ GraphHopper | ◆ OpenCage |
| ◆ Here | ◆ Pelias |
| ◆ HostIp | ◆ Photon |
| ◆ IpInfo | ◆ PickPoint |
| ◆ IpInfoDB | ◆ TomTom |
| ◆ ipstack | ◆ Yandex |
| ◆ LocationIQ | |

Поддержка геолокации в веб-хостингах

Некоторые веб-хостинги предлагают встроенные возможности геолокации, включая такую информацию, как континент посетителя, его страну, штат/регион или город. Эти данные можно учитывать при отображении разного контента в зависимости от местоположения посетителей.

Например, хостинг WP Engine среди прочих своих сервисов предоставляет возможности геолокации (bwawwp.com/wpe-geotarget), которые интегрированы с его фирменной системой кэширования посредством специального плагина для WordPress. WP Engine хранит закешированную информацию о пользователе в бакетах, к которым можно обращаться через заранее заданные переменные PHP и шорткоды WordPress. Это, наверное, один из самых быстрых способов интегрировать геолокационные данные в ваш веб-сайт, но при этом вы должны использовать WP Engine в качестве хостинга.

Сжатие и архивация данных

Можно представить себе несколько ситуаций, в которых требуется программное сжатие и распаковка файлов, но чаще всего приходится выполнять следующие две операции:

- ◆ автоматическая распаковка нескольких ресурсов из одного сжатого ZIP-файла;
- ◆ автоматическая архивация нескольких ресурсов в виде одного сжатого ZIP-файла.

При установке плагинов ядро WordPress выполняет распаковку сжатых файлов. Когда вы загружаете свой собственный плагин или устанавливаете один из тех, что доступны в репозитории WordPress, система находит ZIP-файл и распаковывает его в директорию `wp-content/plugins` вашего сайта.

На сайте AppPresser.com мы активно применяем программную упаковку и распаковку файлов. В нашем конструкторе App Builder UI мы предоставляем поле, с по-

мощью которого пользователи могут загрузить ZIP-архив со всеми графическими ресурсами, которые они хотят использовать локально внутри своего мобильного приложения. Наш серверный PHP-код получает этот архив, распаковывает его и перемещает все файлы в локальную директорию проекта. Когда приложение готово к компиляции, мы автоматически упаковываем все исходные файлы в этой директории и отправляем их сервису PhoneGap Build для дальнейшей сборки. В результате все графические ресурсы, которые изначально были загружены в виде единого ZIP-файла, оказываются доступными локально внутри скомпилированного мобильного приложения.

Когда-то мы занимались переносом одной крупной проприетарной CMS в много-сайтовую сеть WordPress. Сайты, с которых осуществлялась миграция, имели отдельные базы данных MySQL. Нам предоставили учетную запись SFTP, где находились все веб-сайты, требующие переноса, и каждый из них был упакован в отдельный ZIP-архив. В каждом архиве хранилась копия соответствующей БД в виде файла *.sql, а также все исходные файлы и ресурсы сайта. Всего нужно было выполнить миграцию для более чем 8000 сайтов, т. е. от нас требовалось распаковать больше 8000 ZIP-файлов и запустить для каждого из них самописные скрипты миграции, чтобы переместить все данные в WordPress. Нам удалось автоматизировать весь этот процесс: мы циклически проходились по всем архивам в учетной записи SFTP и распаковывали их с помощью PHP ZipArchive (b2wawp.com/php-zip).

Класс PHP ZipArchive предоставляет огромное количество функций, с помощью которых вы можете упаковывать и распаковывать любые файлы, и обычно он входит в стандартную поставку PHP. Файлы можно сжимать либо по одному, либо целыми директориями. Главной функцией для открытия нового ZIP-архива на чтение, запись или изменение является метод ZipArchive::open. В качестве параметра filename он принимает путь к ZIP-файлу, который нужно открыть; вы также можете указать один из четырех режимов с помощью параметра flag:

- ◆ OVERWRITE — перезаписывает файлы в заданном архиве, если они уже существуют.
- ◆ CREATE — создает новый архив, если он еще не существует.
- ◆ EXCL — позволяет проверить, существует ли заданный архив.
- ◆ CHECKCONS — проводит дополнительные проверки целостности архива и в случае неудачи выдает ошибку.

Упаковка в архив отдельных файлов

В следующем примере показан простой код для поштучного добавления файлов в новый ZIP-архив. Как несложно догадаться, вы можете при желании циклически перебирать любые файлы.

Пример

```
<?php
```

```
// Сначала создаем экземпляр ZipArchive  
$zip = new ZipArchive();
```



```
// Используем метод open(), чтобы создать на сервере ZIP-файл
$zip->open('compressed-dir-path/messenlehner-kids.zip', ZipArchive::CREATE);

// Используем метод addFile(), чтобы добавить файл в наш архив
$zip->addFile('some-dir-path/index.html', 'index.html');

// Вы также можете добавлять файлы в поддиректории внутри архива
$zip->addFile('some-dir-path/dalya.png', 'images/dalya.png');
$zip->addFile('some-dir-path/brian.png', 'images/brian.png');
$zip->addFile('some-dir-path/nina.mp4', 'videos/nina.png');

// Вы можете изменить название файла, который упаковывается
$zip->addFile('some-dir-path/CWM.png', 'images/cam.png');
$zip->addFile('some-dir-path/babyA.png', 'images/aksel.png');

// Вызываем метод close(), чтобы завершить процедуру и сохранить новый ZIP-архив
$zip->close();
?>
```

Упаковка нескольких файлов в один архив

ZipArchive предоставляет функции, которые позволяют добавлять в ZIP-архив все содержимое заданной директории, а не отдельные файлы:

- ◆ addGlob() — этот метод позволяет определять файлы заданного типа;
- ◆ addPattern() — этот метод позволяет определять нужные файлы в директории по их именам, используя регулярные выражения.

Далее приведен пример использования методов addGlob() и addPattern() для добавления в ZIP-архив всех изображений в форматах PNG и JPG, а также видеофайлов MP4, находящихся в заданной директории.

Пример

```
<?php

// Сначала создаем экземпляр ZipArchive
$zip = new ZipArchive();

// Используем метод open(), чтобы создать на сервере ZIP-файл
$zip->open('compressed-dir-path/messenlehner-kids.zip', ZipArchive::CREATE);

// Создаем массив параметров архива и добавляем в него путь к директории 'videos'
$options = array('add_path' => 'videos/', 'remove_all_path' => TRUE);

// Берем из заданной директории файлы 'mp4' и добавляем их
// в директорию 'videos' нашего архива
$zip->addGlob('some-dir-path/*.mp4', 0, $options);
```

```
// Добавляем в массив параметров архива и добавляем в него путь к директории
'images'
$options = array('add_path' => 'images/', 'remove_all_path' => TRUE);

// Берем из заданной директории файлы 'jpg' и 'png' и добавляем их
// в директорию 'images' нашего архива
$zip->addGlob('some-other-dir-path/*.{jpg, png}', GLOB_BRACE, $options);

// Для получения любого файла в заданной директории можно также
// использовать регулярные выражения
$zip->addPattern('/\.(?:jpg|png)$/', 'another-dir-path', $options);

// Вызываем метод close(), чтобы завершить работу и сохранить новых ZIP-архив
$zip->close();
?>
```

Извлечение файлов из архива

Для ручной распаковки сжатого архивного файла можно использовать метод `extractTo()`, который принимает два параметра:

- ◆ **destination** — директория, в которую будут распакованы и сохранены файлы из ZIP-архива;
- ◆ **entries** — файлы, которые вы хотите извлечь. Вы можете распаковывать их по одному или указать массив путей к нужным вам файлам. Если же вы хотите извлечь все содержимое архива, можете опустить этот параметр, как показано в следующем примере.

Пример

```
<?php
// Извлечение файлов из ZIP-архива на сервере
$zip = new ZipArchive();
$zip->open('compressed-dir-path/messenlehner-kids.zip',
ZipArchive::CREATE);
// Используем метод extractTo(), чтобы извлечь файлы из ZIP-архива на сервере
// Извлекаем один файл
$zip->extractTo('uncompressed-dir-path/', 'images/aksel.png');
// Извлекаем массив файлов
$zip->extractTo('uncompressed-dir-path/',
array('images/cam.png', 'images/nina.png'));
// Извлекаем все файлы в архиве
$zip->extractTo('uncompressed-dir-path/');
$zip->close();
?>
```

Другие PHP-библиотеки для сжатия и архивации

Для любых операций по упаковке/распаковке файлов WordPress по умолчанию использует ZipArchive. Если быть точным, это библиотека, наличие которой проверяется в первую очередь; если ее нет, WordPress прибегает к другой популярной PHP-библиотеке под названием PclZip. При желании для сжатия и архивации можно применить и другие инструменты:

- ◆ PclZip (bwawwp.com/pclzip);
- ◆ Bzip2 (bwawwp.com/bzip2);
- ◆ LZF (bwawwp.com/lzf);
- ◆ Phar (bwawwp.com/phar);
- ◆ Rar (bwawwp.com/rar);
- ◆ Zlib (bwawwp.com/zlib).

Инструменты для разработки

Далее приведены некоторые PHP-библиотеки, которые могут сделать жизнь разработчиков чуточку проще. Полезных инструментов очень много, и это лишь несколько наших любимых.

PHPUnit

Если вы еще не слышали о *модульном тестировании*, вам следует обратить на него внимание. Это отличный метод регрессивного тестирования кода, который позволяет автоматически находить программные ошибки. Это, в сущности, тестирование кода путем написания другого кода. **PHPUnit** (bwawwp.com/phpunit) позволяет подготовить определенные модульные тесты, которые будут выполняться для отдельных функций в вашем проекте, проверяя, работают ли они так, как задумано. Важность модульного тестирования сложно преувеличить, особенно когда написанием кода занимается несколько человек. Выполнение модульных тестов перед развертыванием каких-либо обновлений в промышленной среде поможет сохранить душевный покой как вам, так и вашим клиентам.



За внедрение модульного тестирования в процесс разработки можно и нужно требовать надбавку.

phpDocumentor

Любой приличный разработчик знает, что он всегда должен предоставлять хорошие комментарии и дополнительную документацию, чтобы коллегам было проще разобраться в его коде. Это также полезно при просмотре своего старого кода, написание которого вы с трудом припоминаете. По меньшей мере, вы всегда должны составлять хорошие комментарии. **phpDocumentor** (bwawwp.com/phpdoco) — это

отличный инструмент для автоматического создания документации на основе комментариев в коде.

Faker

Faker (bwwawp.com/faker) — это прекрасная PHP-библиотека для автоматической генерации фиктивного контента, которым можно наполнять базу данных, создавать ненастоящих пользователей WordPress, отображать шаблонный текст на своем веб-сайте, еще когда вам нечего публиковать, генерировать XML-потoki, наполнять данными созданные вами конечные точки API-интерфейса, проводить модульное тестирование и выполнять любые другие задачи на ваше усмотрение. Для генерации фиктивных данных можно использовать следующие средства форматирования (но вы также можете создавать свои собственные, взяв за основу готовые функции, определенные в классе `Base`):

- | | |
|-----------------------|------------------------|
| ◆ Base | ◆ Payment |
| ◆ Lorem Ipsum Text | ◆ Color |
| ◆ Person | ◆ File |
| ◆ Address | ◆ Image |
| ◆ Phone Number | ◆ Uuid |
| ◆ Company Real | ◆ Barcode |
| ◆ Text Date and Time | ◆ Miscellaneous Biased |
| ◆ Internet User Agent | ◆ Html Lorem |

Goutte: веб-скрапер, написанный на PHP

Переносите веб-сайт на WordPress, но не имеете доступа к источнику его данных? Хотите получить данные из нескольких веб-сайтов, у которых нет каналов данных? Веб-скрапинг может стать отличным решением для некоторых из ваших проблем, однако программный поиск нужной информации по заданному шаблону внутри веб-страницы (с надеждой на хоть какую-то однородность) с последующим анализом HTML-разметки и извлечением из нее подходящих фрагментов может отнять очень много времени. Если вы занимаетесь скрапингом контента с нескольких веб-страниц, эта задача может оказаться крайне утомительной.

Goutte (bwwawp.com/goutte) — это потрясающий инструмент для извлечения практически любых данных из любой HTML-разметки, который также пригоден для анализа XML. Goutte позволяет легко анализировать HTTP-ответ с помощью готовых методов, которые помогут вам найти и извлечь любые HTML-элементы. Вот некоторые крутые трюки, на которые способен этот инструмент:

- ◆ извлечение ссылок из контента;
- ◆ извлечение изображений из контента;
- ◆ извлечение заголовков из тегов `<header>`;

- ◆ извлечение содержимого из тегов <div> с определенным ID или именем класса;
- ◆ извлечение соседних, родительских или дочерних элементов в упорядоченном или неупорядоченном списке;
- ◆ извлечение содержимого любого HTML-тега;
- ◆ извлечение любых узлов из XML;
- ◆ автоматический переход по ссылкам;
- ◆ автоматическая отправка веб-форм;
- ◆ автоматический вход на сайт (если у вас есть подходящие имя пользователя и пароль).

Whoops: удобочитаемые сообщения об ошибках в PHP

Whoops (bwawwp.com/whoops) — это PHP-библиотека, доступная в виде пакета Composer, которая позволяет обрабатывать ошибки и исключения в вашем коде более элегантным образом. Вы можете снабжать PHP-ошибки какой угодно информацией, расширяя стандартные сообщения, в которых иногда сложно разобраться. Это прекрасный инструмент для обработки и вывода ошибок, который поможет вам в поиске и устранении неполадок.

Внешние API-интерфейсы и веб-сервисы

API-интерфейс может быть частью приложения, исходный код которого вам изначально доступен (как, например, в случае с различными API-интерфейсами WordPress, рассмотренными в этой книге). Но он также может играть роль механизма для работы со сторонними приложениями и сервисами. Многие веб-проекты предоставляют какого-то рода API-интерфейс для чтения или изменения своих данных из других приложений.

Давайте рассмотрим API-интерфейсы наших любимых веб-сервисов, которые можно с легкостью интегрировать в WordPress.

Elasticsearch

Это связка из Elasticsearch (bwawwp.com/elastic-search), Logstash, Kibana и Beats (известная как *ELK Stack*). Надежно и безопасно извлекает из любого источника данные произвольного формата, после чего анализирует и визуализирует их в реальном времени, позволяя выполнять по ним поиск.

ElasticPress by 10up

ElasticPress by 10up (bwawwp.com/elasticpress) автоматически интегрируется с Elasticsearch для индексации базы данных WordPress, предоставляя развитые поисковые функции, которые можно использовать в своем веб-приложении.

Google Vision

Невероятно крутой, но в то же время немного пугающий проект. Google Vision API (bwawwp.com/google-vision) использует машинное обучение для анализа заданных изображений и пытается определить их содержимое с той или иной вероятностью.

Google Maps

Все знают, что такое Google Maps, и мы уверены в том, что вам уже попадались плагины, которые интегрируют этот сервис в WordPress. Google предлагает API-интерфейс на основе JavaScript (bwawwp.com/googlemaps-api) для взаимодействия со своими картами; с его помощью можно создать что угодно — от простой карты какой-то определенной местности до пользовательских карт с маркерами, обозначающими различные посты, сделанные в WordPress из определенного СРТ. Возможности практически безграничные!

Помимо функций, доступных через Google Maps JavaScript API, компания Google предоставляет и другие API-сервисы, относящиеся к своим картам.

Directions

Этот сервис подойдет для прокладки маршрутов от одной точки к другой с подробной схемой проезда. При формировании запроса к этому API-интерфейсу можно указать начальную и конечную точки вашего маршрута. Вы также можете выбрать режим передвижения: на автомобиле (по умолчанию), на велосипеде, транзитом или пешком.

Distance Matrix

Google Distance Matrix API — это сервис, который вычисляет длину и продолжительность маршрутов для матрицы начальных и конечных координат. Возвращаемая информация основана на рекомендуемых маршрутах, предлагаемых сервисом Google Maps API; для каждой пары вычисляется продолжительность путешествия и расстояние.

Elevation

Этот API-интерфейс позволяет запрашивать для заданной точки данные о высоте над уровнем моря.

Geocoding

С помощью этого API-интерфейса можно запрашивать геолокационные данные, такие как широта и долгота, для заданного адреса. Возможно также обратное геокодирование, чтобы получить ближайший адрес для заданных широты и долготы.

Сервис Street View

Это API-интерфейс позволяет взаимодействовать с Google Street View. Это по-настоящему мощный инструмент, с помощью которого можно получить доступ ко всем фотографиям и местоположениям, доступным в Google Street View.

Плагины Google Maps для WordPress

Далее перечислено несколько полезных плагинов для WordPress, в которых уже используется сервис Google Maps:

- ♦ *WP Google Maps* (bwawwp.com/wp-google-maps) — один из популярнейших плагинов для работы с Google Maps. Позволяет с легкостью создавать карты в постах WordPress с использованием коротких кодов.
- ♦ *Gutenberg Map Block for Google Maps* (bwawwp.com/map-block-gutenberg) — если вы используете Gutenberg и хотите предоставить своим пользователям блок для управления Google Maps, этот плагин то, что вам нужно.
- ♦ *gPano* (bwawwp.com/gpano) — простой в использовании плагин для WordPress, который позволяет встраивать в посты и страницы панорамные изображения из Google Street View.
- ♦ *WP Store Locator* (bwawwp.com/wp-store-locator) — простой в применении плагин для WordPress, который позволяет добавлять на карту географические местоположения с пользовательскими маркерами и информацией. Местоположение можно искать по названию, метаданным или удаленности от заданной точки. Этот плагин также задействует сервис Google Maps Directions для прокладывания маршрутов к заданному местоположению.

Google Translate

Если вам нужно автоматически переводить отдельные фрагменты своего контента на разные языки, то можно воспользоваться Translation API (bwawwp.com/googletranslate) от Google. Существует немало плагинов для WordPress, которые работают с этим API-интерфейсом, но вы также можете использовать его напрямую, чтобы переводить различный текст на лету. Кроме того, компания Google внедрила в этот сервис функцию машинного обучения под названием *AutoML Translation*. Это платный API-интерфейс, но он вполне может себя окупить, если вам действительно необходимо автоматически переводить разные части вашего контента.

Translate WordPress with GTranslate (bwawwp.com/gtranslate) — это замечательный плагин, который интегрируется в WordPress и демонстрирует некоторые возможности, предоставляемые сервисом Google Translate.

Twilio

Хотите отправлять гибкие SMS-оповещения подписчикам своего веб-приложения? Возможно, вашему проекту нужен дополнительный слой безопасности, и вы хотите

подтверждать пользовательские учетные записи с помощью случайных текстовых кодов активации, отправляемых на мобильные телефоны. Twilio (bwawwp.com/twilio) предлагает отличный веб-сервис для обмена SMS-сообщениями между сотовыми телефонами и вашей учетной записью (одной или несколькими).

Представьте, что руководству школы необходимы телефонные номера родителей, чтобы иметь возможность рассылать им оповещения с важной информацией — например, когда их ребенок прогуливает уроки. Первым делом следует загрузить с GitHub (bwawwp.com/twilio-php) PHP-библиотеку Twilio. Если вы разрабатываете собственный плагин, скопируйте ее содержимое вместе с файлом Twilio.php в директорию `lib` своего проекта. Ваша структура директорий должна иметь примерно такой вид: `/wp-content/plugins/<ваш-плагин>/lib/Twilio`.



Мы рекомендуем работать с Twilio API по SSL.

Другие популярные API-интерфейсы

Ранее мы перечислили несколько популярных PHP-библиотек, веб-сервисов и API-интерфейсов, которые подойдут для решения ваших задач. Но это лишь верхушка айсберга. В Интернете можно найти огромное количество инструментов, как проприетарных, так и доступных публично.

Помните, не стоит изобретать велосипед. Прежде чем что-то создавать, проверьте уже готовые ресурсы. Используйте данные из внешних источников, интегрируйте собственные данные с внешними социальными сетями и каталогами. Работайте умнее, а не сильнее!

Перечислим некоторые другие популярные веб-сервисы, с которыми можно поэкспериментировать:

- ◆ *Google APIs Explorer* (bwawwp.com/google-apis) — список веб-сервисов Google с соответствующими API-интерфейсами, включая YouTube, Drive, Calendar, Gmail, Analytics, Hangouts и Firebase.
- ◆ *Instagram Graph API* (bwawwp.com/instagram-api) — этот API-интерфейс позволяет создавать приложения, с помощью которых пользователи с учетными записями уровня Business и Creator могут управлять своими фотографиями, видеороликами, историями, альбомами, комментариями и хэштегами.
- ◆ *Salesforce API Explorer* (bwawwp.com/salesforce-apis) — список доступных сервисов и API-интерфейсов Salesforce, таких как Marketing Cloud, Einstein Prediction, Force.com и Data.com.
- ◆ *Flickr API* (bwawwp.com/flickr-api) — API-интерфейс для программного управления фотографиями и контентом в Flickr.
- ◆ *eBay API* (bwawwp.com/ebay-api) — eBay предоставляет несколько API-интерфейсов для поиска, покупки, продажи, согласования цены и практически всего остального, чем могут заниматься покупатель и продавец.

- ◆ *Dropbox API (bwawwp.com/dropbox-api)* — управляйте своими учетными записями Dropbox программным образом. Внедряйте в свои приложения такие возможности Dropbox, как хранение, совместное использование, предварительный просмотр и поиск файлов.
- ◆ *LinkedIn API (bwawwp.com/linkedin-api)* — API-интерфейсы для программного поиска и распространения контента этой платформы.
- ◆ *MailChimp API (bwawwp.com/mailchimp-api)* — для синхронизации электронных переписок, организации рассылок, управления аудиториями и автоматизации рабочих процессов.
- ◆ *Constant Contact API (bwawwp.com/cc-api)* — управляйте программным образом учетными записями Constant Contact: извлекайте, создавайте и обновляйте свои контакты.

Практически любой крупный веб-сервис позволяет работать со своими данными с помощью какого-нибудь API-интерфейса.

Миграция

Ах миграция, миграция, миграция... с чего же начать? Давайте вернемся в самое начало — к моей первой крупномасштабной миграции. Во времена моей юности, еще до знакомства с WordPress, я работал в команде Marine Logistics, базировавшейся в Кэмп-Лежен, Северная Каролина. Проходя службу в Корпусе морской пехоты США, я научился создавать крупномасштабные базы данных с нуля с последующим их обновлением и обслуживанием. Я освоил все тонкости SQL (Structured Query Language — структурированного языка запросов) и отличия в синтаксисе при работе с разными БД. Мы в основном использовали Microsoft SQL Server, но в некоторых крупных проектах применялись продукты Oracle. У большинства баз данных, которые я создал и над которыми работал, был клиентский веб-интерфейс, написанный на ColdFusion, Classic ASP или ASP.NET, и со временем я сильно наловчился писать код на этих языках программирования. Это позволяло мне писать код для импорта и экспорта любых данных, которые мне требовались, для любых хранилищ. Одна из крупнейших веб-систем, с которыми мне приходилось иметь дело во время службы (и по сей день), была способна отслеживать огромное количество оснащения и техники всевозможных типов, от танков до зубных щеток. Эта логистическая платформа была разработана совместно с гражданскими подрядчиками для замены мейнфрейма, который морская пехота задействовала в те времена для учета материального обеспечения развертываемых подразделений. Моей задачей был перенос огромных объемов данных из мейнфрейма в базу данных Oracle, которую использовало новое веб-приложение. Мне было приказано отложить все дела и осуществить эту миграцию, и именно этим я занимался на протяжении нескольких месяцев, пока все данные не были как следует импортированы. И хотя дело было еще до появления WordPress, большинство уроков, которые я усвоил более 15 лет назад, работая над этим проектом миграции для морской пехоты, до сих пор актуальны.

Давайте поговорим о процессе миграции и о том, как перенести все в WordPress. Когда мы слышим "миграция на WordPress" без каких-либо уточнений, на ум приходят следующие две вещи: миграция между серверами и миграция между платформами. В зависимости от поставленной перед вами задачи вам придется освоить один или оба эти процесса. Что бы от вас ни требовалось, существуют инструменты и плагины для WordPress, которые помогут сделать вашу работу менее утомительной. О них и пойдет речь дальше. Но сначала давайте обсудим виды миграции, с которыми вы можете столкнуться.

Миграция между серверами

Миграция сервера — это перенос своего сайта на основе WordPress с другого компьютера или хостинга. Этим часто приходится заниматься разработчикам, которые запускают новые веб-сайты для своих клиентов или переносят существующие сайты на другой хостинг. Но все сводится к тому, что готовый проект должен куда-то "мигрировать". И этот процесс в основном состоит из одних и тех же шагов.

1. Сделайте резервную копию!
2. Переместите все свои исходные файлы на новый сервер.
3. Экспортируйте и импортируйте свою базу данных.
4. При необходимости поменяйте настройки DNS.
5. Обновите в базе данных все URL-адреса, которые этого требуют.
6. Убедитесь в том, что ничего не "сломалось".

Доступные плагины для миграции

Duplicator WordPress Migration Plugin (bwawwp.com/duplicator) — надежный плагин для переноса сайтов на основе WordPress.

WP Migrate DB Pro (bwawwp.com/wpmdbp) — это один из наших любимых плагинов для импорта и экспорта данных в WordPress и из него. Он поддерживает простые и сложные процессы миграции. С его помощью можно с легкостью перенести таблицы базы данных с одного сервера на другой. Этот плагин также отлично подходит для работы с эксплуатационными, промежуточными и локальными окружениями, а также со средами разработки. Вы можете активировать и сконфигурировать его во все своих средах и легко перемещать данные с помощью графического интерфейса. У этого проекта есть несколько дополнительных плагинов, но вы, конечно же, можете создавать свои собственные. Среди самых примечательных расширений можно выделить CLI Add-on и Multisite Tools Add-on.

Средства миграции сайтов от WP Engine

Некоторые хостинговые компании, такие как WP Engine, предлагают средства миграции, которые максимально упрощают переход на их платформы. Компания пытается избавить вас от необходимости ручного переноса вашего веб-сайта или веб-приложения.

Миграция между платформами

Переходом на WordPress с другой платформы занимаются многие, но это может потребовать значительных усилий в зависимости от того, сколько у вас данных и насколько хорошо они структурированы. На WordPress можно мигрировать как с других открытых платформ вроде Drupal или Joomla, так и с самописных проприетарных систем. В любом случае ваша задача сводится к автоматизации импорта данных в WordPress. Следуйте инструкциям, приведенным далее, и вы сможете не только переместить свои данные, но и не поседеть в процессе.

Куда переносятся данные

Доставка контента в WordPress — это лишь полдела! Мы всегда были большими любителями написания скриптов для автоматического наполнения баз данных. Главное в этом деле — доставить информацию туда, где ей место, и в нужном формате, и неважно, какая задача перед вами стоит: написать простую веб-форму для ручного ввода данных в БД или извлечь миллионы записей из одной БД и импортировать их в другую. Очень важно, чтобы вы четко себе представляли, как организован весь ваш контент и каким образом он структурирован в вашей БД. Понимание структуры базы данных WordPress и владение вспомогательными функциями, которые входят в ядро этой системы и позволяют взаимодействовать с БД, является незаменимым в процессе миграции. Если база данных WordPress кажется вам слегка загадочной и вы не вполне понимаете, как она устроена, возможно, вам не следовало пропускать *главу 2*!

Откуда переносятся данные

Итак, мы уже знаем, что наши данные должны попасть в WordPress, но их источником может быть что угодно: другой сайт на основе WordPress, иная платформа, написанная на PHP и MySQL, такая как Drupal или Joomla, проприетарная структура таблиц, хранящаяся в MS SQL Server, Oracle или БД любого другого типа; это могут быть данные в формате XML или JSON, извлекаемые из API-интерфейса, или набор CSV-файлов или даже статических HTML-документов. Этот список можно продолжать до бесконечности: данные, которые вам нужно перенести в WordPress, могут находиться в любом источнике и выглядеть как угодно. Но вас в первую очередь должно интересовать не то, какую информацию и откуда необходимо получить, а то, как она структурирована и каким образом ее можно адаптировать к WordPress. Мы имели дело с миграцией практически любых данных, которые только можно себе представить. Поэтому далее перечислим некоторые источники информации, с которыми нам приходилось работать.

Содержимое баз данных

Как показывает наш опыт, в роли внешнего источника данных при миграции на WordPress чаще всего выступают другие реляционные БД. То, насколько сложным

окажется процесс перехода, зависит от движка базы данных, структуры таблиц и размера их содержимого:

- ♦ Структуры таблиц в MySQL. MySQL — это свободная и очень популярная БД, которая чаще других применяется в открытых системах управления контентом и системах веб-публикации на языке PHP, таких как WordPress, Drupal, Joomla!, Magento, ExpressionEngine, SilverStripe и MediaWiki.
- ♦ Структура таблиц в Microsoft SQL Server. SQL Server — это закрытая, но весьма популярная БД. Ее активно используют как в простых веб-сайтах, так и корпоративных приложениях. В большинстве случаев у SQL Server есть клиентский веб-интерфейс, написанный на .NET.
- ♦ Структура таблиц в Oracle. БД Oracle обычно целесообразна для крупных проектов с огромными объемами данных (миллионы записей). Хотя мы уже работали с несколькими клиентами, которые использовали Oracle, и можем сказать, что это было явным перебором. На самом деле они искали более элегантное, практичное и масштабируемое решение, такое как свежий пользовательский интерфейс на основе WordPress и MySQL.
- ♦ Другие движки баз данных. Существуют другие движки баз данных, и со всеми ними мы уже работали. Речь идет о PostgreSQL, DB2, mongoDB, Microsoft Access, SQLite, Sybase и т.д.

Данные из файлов

Иногда у нас нет прямого доступа к базе данных. Возможно, вся информация вашего веб-приложения хранится в файлах на сервере, или же ваша система позволяет экспортировать только файлы. Как бы то ни было, вам нужно понимать, как структурировано содержимое этих файлов. Перечислим некоторые типичные файловые форматы для хранения данных:

- ♦ Comma-separated values (CSV) — файлы с разделителями бывают разных типов, но самым распространенным является CSV.
- ♦ Extensible markup language (XML) — часто используется для обмена данными в Интернете. У XML есть несколько разновидностей: RSS, Atom, SOAP, XHTML и т.д.
- ♦ JavaScript Object Notation (JSON) — файлы, в которые сохраняются данные этого формата, обычно имеют расширение txt. JSON повсеместно применяется для обмена данными и во многом заменяет XML.
- ♦ Hypertext markup language (HTML) — если вы дочитали до этого места, вам, скорее всего, не нужно объяснять, что такое HTML.

Данные из внешних API-интерфейсов

API-интерфейсы — это крайне распространенный механизм переноса данных из одной веб-системы в другую. В этой главе мы уже упоминали несколько популярных веб-сервисов и API-интерфейсов.

Скрапинг веб-страниц

Если ни один из перечисленных ранее способов не работает и вы не можете получить доступ к своим данным, всегда остается последний вариант: веб-скрапинг (извлечение контента из активного сайта). Сложность этого процесса зависит от того, насколько хорошо структурирован HTML-код в ваших веб-страницах. Для скрапинга требуется однородность и последовательность.

Каким бы ни был источник ваших данных, идея всегда одна и та же: проанализировать информацию, перебрать ее циклически и поместить в нужное место. Звучит слишком уж просто, но в большинстве случаев все сводится именно к этому. Главное — хорошо владеть WordPress и иметь четкое представление о своих источниках данных.

Руководство по привязке данных

Это один из важнейших аспектов любой миграции, даже самой мелкой. Привязка данных из вашего источника к тому хранилищу, в которое они должны попасть, сэкономит вам уйму времени в долгосрочной перспективе. Мы делаем это в каждом проекте независимо от его размера, и вам, скорее всего, стоит последовать нашему примеру. На самом деле наше руководство по привязке данных довольно простое. Мы советуем использовать электронные таблицы Google, чтобы члены команды могли легко наладить совместную работу в режиме реального времени. Вначале определите типы исходного контента, такие как посты, комментарии, пользователи и т. д. В левой части таблицы перечислите каждый тип в столбик вместе с любыми полями метаданных, которые он содержит. Укажите рядом с каждым полем тип данных (например, строка, целое число или булево значение). Вслед за типом можно предоставить пример реальных данных. Дальше могут следовать заметки или подробные комментарии о том или ином поле. Мы советуем проделать это для всех типов контента, которые вы переносите в WordPress.

Создайте справа новый столбец под названием "Поля WordPress" и начните сопоставлять каждое исходное поле с конечным. Некоторые привязки довольно очевидны, как в случае с заголовком поста. Некоторые исходные метаданные превратятся в метатеги WordPress, в этом случае следует добавить соответствующие ключи. Если вы занимаетесь разработкой новой темы оформления или используете плагин для WordPress с заранее определенными метаключами, не забудьте указать все нужные вам метатеги. Если вы выполняете миграцию для своих клиентов, то будет полезно пройти с ними вместе по намеченному процессу привязки данных и получить их одобрение. Убедитесь в том, что ваши клиенты и все, кто вовлечен в процесс миграции, достигли полного взаимопонимания относительно того, куда должны попасть те или иные данные. Если пропустить составление этого руководства, вам, возможно, придется проводить многократные совещания со своими клиентами и затем писать дополнительные скрипты для исправления данных или вносить правки в свой основной скрипт миграции и запускать его заново. В зависимости от объемов импортируемых данных повторное выполнение процесса миграции с нуля займет много времени, которого у вас может не быть. Если коротко, то обязательно создайте руководство по привязке данных, в долгосрочной перспективе вы будете рады этому решению.

Взгляд в будущее

Первое издание этой книги было опубликовано в 2014 году. За пять лет, которые разделяют первое и второе издания, много чего изменилось. И в следующие пять лет ожидаются большие изменения.

В этой заключительной главе мы кратко пройдемся по некоторым из крупных изменений, которые претерпел проект WordPress за последние годы. Если вы дочитали до этого места, то уже должны хорошо ориентироваться в этих темах.

Мы также сделаем несколько прогнозов относительно будущего WordPress. Некоторые из них могут и не сбыться, но в целом это те аспекты, на которые стоит обращать внимание в следующие несколько лет.

Оглядываясь назад

Версия WordPress 3.0 была выпущена в июне 2010 года, и в ней впервые появилась поддержка CPT (подробней об этом в *главе 5*). Это был последний шаг на пути к переходу от блоговой платформы к системе управления контентом. Добавление CPT позволило разработчикам вроде нас применять WordPress в качестве фреймворка для создания приложений.

Выпуск WordPress версии 4.7 состоялся в декабре 2016 года и принес поддержку конечных точек REST API для постов, комментариев, терминов, пользователей, метатегов и настроек (REST API подробно рассматривается в *главе 10*). Плагин, который предоставляет такие возможности, был доступен еще за несколько лет до этого выпуска, однако его добавление в ядро WordPress стало сигналом о том, что REST API теперь официально рекомендуется использовать в разработке.

В версии WordPress 5.0, выпущенной в декабре 2018-го, появился новый редактор блоков Gutenberg (подробней о Gutenberg, блоках и CPT читайте в *главе 11*). Это новый инструмент, и, прежде чем сайты, плагины и темы оформления начнут пользоваться его преимуществами, должно пройти какое-то время. Тем не менее редактор блоков вводит новую парадигму управления контентом, разметкой и другими параметрами сайта. Проект Gutenberg также стал примером практического применения современного стека разработки на JavaScript, проиллюстрировав работу с такими инструментами, как Node.js, npm, React, webpack и Git. Вместе с плагинами, внедряющими новые возможности, он задает общие стандарты разработки для WordPress.

REST API

Полноценная поддержка REST API появилась еще в WordPress версии 4.7, но на то, чтобы разработчики и стеки разработки начали использовать весь потенциал API-интерфейсов в своих проектах, ушло еще несколько лет. Теперь REST API применяется в различных участках ядра WordPress, а также во многих важных плагинах. Со временем этот аспект платформы WordPress будет становиться все более важным.

Плагины WordPress будут уделять больше внимания API-интерфейсам

На сегодня во многих новых проектах API-интерфейсы выходят на передний план. Вначале отдельно от клиентской части создаются API-интерфейсы, необходимые приложению. Затем выбирается стек технологий, который лучше всего подходит для клиентской части. Это упрощает обслуживание слоя данных вашего приложения и добавление новых представлений. Один и тот же контент в WordPress может использоваться в веб-сайте, мобильном приложении, почтовой рассылке и любой другой системе, которой нужны эти данные.

В таких плагинах, как WooCommerce и BuddyPress, почти все возможности покрыты API-интерфейсами. Это упрощает интеграцию их данных с приложениями, которые вы подключаете к своему сайту на основе WordPress.

"Обезглавленные" версии WordPress

Все чаще можно встретить проекты, в которых WordPress служит исключительно в качестве хранилища данных для более крупных веб-приложений, которые обращаются к этому хранилищу через REST API. Версии WordPress без встроенных тем оформления и клиентской части, а только с панелью администрирования, называются "обезглавленными".

Статические генераторы сайтов вроде GatsbyJS позволяют развертывать статические сайты, которые отличаются высокой безопасностью, скоростью работы и относительной дешевизной. Вы можете настроить GatsbyJS так, чтобы источником данных являлся WordPress, при этом последний даже необязательно делать доступным онлайн — он может работать в локальной среде.

Обезглавленный WordPress применяется даже в динамических сайтах. Динамические участки веб-страниц, написанные на JavaScript с учетом поставленных требований, работают на клиентской стороне и задействуют сторонние сервисы или API-интерфейсы, размещенные на их собственных серверах.

Сейчас такая компонентная разработка пользуется популярностью. У распределения приложений между несколькими разными серверами и сервисами есть свои плюсы и минусы.

Положительной стороной этого подхода является то, что каждый сервис, который применяется в приложении, оптимизирован под определенную задачу. Если один

компонент выйдет из строя, приложение может обойтись без него какое-то время. Если у одного компонента возникли проблемы с безопасностью, они могут не повлиять на остальное приложение. Кроме того, замена одного компонента должна потребовать меньше усилий, чем обновление целой платформы.

Недостатком использования нескольких сервисов в своем приложении является то, что вам придется потратить дополнительное время на изучения новых платформ и решение проблем с интеграцией. Некоторые сервисы плохо уживаются вместе, и вам, возможно, придется хорошенько подумать о том, как их "подружить".

В разработке ПО существует своеобразный цикл, в котором приложения рекомендуется делать то автономными и самодостаточными, то распределенными между несколькими сервисами. Со временем отдельные инструменты, которые работают совместно, объединяются в хостинговые пакеты, интегрированные среды разработки или управляемые контейнеры. Это позволяет нивелировать некоторые проблемы с интеграцией и упростить применение определенных наборов инструментов. Затем еще через какое-то время эти наборы начинают считать едиными платформами. Когда вы регистрируете сайт с поддержкой WordPress, вам обычно не нужно думать о том, какие сервер, база данных, PHP-модули и другие инструменты потребуются для его работы. В будущем появятся аналогичные сервисы, в которых приложения на основе WordPress можно будет устанавливать одним щелчком мыши, но уже с другим набором технологий, размещенным на других серверах.

GraphQL

GraphQL — это язык запросов для API-интерфейсов и потенциальная замена REST API везде, включая WordPress. В *главе 10* мы потратили много времени на описание и восхваление REST API. Чем же GraphQL лучше?

Традиционно API-интерфейс призван определять конечные точки, включая информацию о том, какие данные должны передаваться в запросе и возвращаться в ответе. В GraphQL сервис сам описывает запрос и формат, в котором он хочет получить данные. Это означает, что для публикации и обслуживания API-интерфейсов требуется меньше усилий, а их использование другими сервисами становится более гибким.

Применение REST API подобно вызову функций в PHP. Вы получаете заранее заданные параметры (хотя некоторые из них необязательные), и данные обычно возвращаются в одном и том же формате. Язык GraphQL больше похож на SQL-запросы. Вы можете извлекать любые доступные данные, объединять их так, как вам хочется, определять значения, которые вы хотите вернуть.

Подробнее о GraphQL можно почитать на сайте graphql.org или в книге Алекса Бэнкса и Иви Порселло *Learning GraphQL* (O'Reilly). Поддержку GraphQL в сайте на основе WordPress можно включить с помощью плагина WP GraphQL. Чтобы узнать больше, посетите www.wpgraphql.com.

Gutenberg

С момента появления нового редактора блоков в WordPress 5.0 наблюдается стремительная разработка различных блоков и сценариев их применения — как в ядре WordPress, так и в других проектах. В этом разделе мы дадим наши прогнозы касательно Gutenberg и редактора блоков.

Интерфейс администратора перейдет на React/Gutenberg

Все настройки, связанные с визуальным отображением контента на клиентской стороне вашего веб-приложения, так и просят, чтобы их реализовали в виде блоков. Перевод виджетов и меню навигации на блоки является частью второго этапа проекта Gutenberg.

Переписывание других страниц администрирования с использованием React и API-вызовов может существенно повысить производительность и удобство работы с панелью управления WordPress. Уже существуют некоторые прототипы, которые иллюстрируют, как представление WordPress List Tables можно перенести на React. Их можно найти в репозитории GitHub под названием New List Tables (oreil.ly/yjXaE).

Gutenberg будет применяться для редактирования контента на клиентской стороне WordPress

Уже сейчас разработчики ядра WordPress и создатели блоков пытаются максимально приблизить редактор блоков к тому, как контент отображается на клиентской стороне. Поскольку в редактор блоков теперь переходят такие элементы интерфейса, как меню, боковые панели и другие части разметки веб-страницы, вы фактически сможете изменять в нем любые аспекты дизайна вашего сайта.

Существуют некоторые разногласия относительно того, каким должен быть опыт взаимодействия при редактировании клиентского контента в WordPress. Но в то же время переключение на панель управления может стать настолько быстрым и прозрачным, что текущий подход к редактированию станет "приемлемым" для многих людей, которые желают изменять клиентский контент. И хотя официальная "функция клиентского редактирования" может появиться не сразу, этот пробел заполнят плагины или что-то другое.

Шаблоны блоков заменят темы оформления

В WordPress темы оформления определяют внешний вид и поведение вашего веб-сайта за счет выбора цветов, шрифтов и разметки. Большинство тем позволяют гибко управлять цветами и шрифтами, а разметку теперь можно изменять с помощью шаблонов блоков.

Это явное упрощение дизайна веб-сайта, ведь для приличного оформления помимо цветов, шрифтов и разметки требуются еще и тонкие художественные элементы.

Но в то же время именно такой гибкости пользователи привыкли ожидать при выборе тем в WordPress.

Разработчики любой приличной темы должны позаботиться о том, чтобы ее дизайн хорошо сочетался с популярными блоками и шаблонами блоков.

Долгое время развитые возможности тем оформления помечали как "территория плагинов". Ключевая идея состояла в том, что пользователь должен иметь возможность менять свои темы без потери какого-либо контента или функциональности веб-сайта. Все, что нарушает этот принцип, следует выносить в отдельный плагин. Темы оформления, размещенные в репозитории WordPress.org, полностью соблюдают это правило, и разработчики сторонних платных тем пытаются следовать их примеру, но при этом предоставляют поддержку определенных сценариев использования.

Многие магазины, продающие темы, фактически превратились в каталоги премиальных плагинов, которые позволяют этим темам раскрыть весь свой потенциал. Часто "территорию плагинов" в таких темах называют "территорией блоков".

Темы оформления никуда не денутся. Они по-прежнему будут определять внешний вид вашего веб-сайта, но со временем управление заголовками, колоннитулами, боковыми панелями и другими элементами разметки перейдет к редактору блоков.

Блоки заменяют плагины

Фраза "для этого уже есть плагин" — популярный мотив в сообществе WordPress. В одном только репозитории WordPress.org находится более 50 000 плагинов. Очень может быть, что практически для любой функции или интеграции, которые вы ищете, существует подходящий плагин. Вам просто нужно его найти.

Но поскольку редактор блоков все активнее применяется для построения сайтов на основе WordPress, пользователи будут в первую очередь искать блоки, которые удовлетворяют их нуждам. Многие из этих блоков будут реализованы в виде плагинов, но их также можно добавлять путем загрузки файлов JavaScript с общего сервера.

Плагины никуда не денутся, но их все чаще будут рассматривать как нечто, связанное с блоками. Пользователи будут искать не плагин подписки, а блок подписки. Существующие плагины способны добавлять новые блоки¹, но плагины, изначально построенные из блоков, будут более доступными для пользователей и смогут быстрее поспевать за новым технологическим стеком и парадигмами разработки, стандартизованными проектом Gutenberg.

Доля WordPress на рынке будет колебаться

WordPress используется примерно в каждом третьем веб-сайте, доступном в Интернете. В период между 2011 и 2019 годами рыночная доля этой платформы выросла с 13,1% до 34% от всех сайтов. И статистически эти темпы роста не демонстрируют заметного падения.

¹ В Paid Memberships Pro эта возможность появилась вскоре после выхода редактора блоков.

С точки зрения конечного пользователя WordPress конкурирует с такими сервисами, как Squarespace и Shopify. Хостинги с поддержкой WordPress наладили простой процесс создания новых веб-сайтов, а новый редактор на основе блоков упростил процесс разметки контента. Эти улучшения означают, что WordPress продолжит успешно конкурировать с удаленными управляемыми решениями.

Вертикальная консолидация организаций, разрабатывающих WordPress, а также тот факт, что хостинг-провайдеры покупают компании, которые создают плагины и темы оформления, поможет в дальнейшем оптимизировать процесс создания веб-сайтов на основе данной платформы.

Количество веб-сайтов, полностью основанных на WordPress, может стабилизироваться или даже немного уменьшиться, но доля отдельных проектов, которые используют WordPress как часть более обширного стека технологий, продолжит расти.

WordPress станет более популярной платформой для мобильной разработки

REST API, стандарты для прогрессивных веб-приложений и такие инструменты, как AppPresser, делают мобильную разработку с помощью WordPress проще простого. Если вы пишете веб-приложение с использованием React и "обезглавленного" экземпляра WordPress, тот же экземпляр будет логично применять и в мобильном клиенте на React Native.

WordPress будет оставаться хорошим выбором для разработки любого рода приложений

Спустя пять лет платформа WordPress станет еще лучше, будет применяться в еще большем количестве интернет-проектов. Улучшатся API-интерфейсы и дополнительные инструменты, хостинги и другие сервисы. У этой книги выйдет еще одно или два издания.

Мы в восторге от той замечательной работы, которую проделали разработчики под WordPress, включая вас. Если вам удастся создать с помощью WordPress нечто потрясающее, дайте нам об этом знать. Пишите нам в "Твиттер", @BWAwp (twitter.com/bwawwp), или свяжитесь с нами на сайте bwawwp.com.

Об авторах

Брайан Мессенленер начал свою карьеру разработчика программного обеспечения в Корпусе морской пехоты США в 2000 году. Он является сооснователем AlphaWeb.com, AppPresser.com и SchoolPresser.com — интернет-компаний, которые специализируются на разработке веб-приложений и мобильных решений на основе WordPress. С 2008 года Брайан работает с WordPress в качестве специалиста, разрабатывая веб-сайты и мобильные решения для таких клиентов, как журнал TIME, NBC, Microsoft, Discovery Channel, Constant Contact, Uber, Campbell's Soup, NEB, Starbucks, YMCA, государственные школы Ньюарка (Нью-Джерси), Служба национальных парков США и другие. Брайана можно найти в "Твиттере" по псевдониму **@bmess**.

Джейсон Коулман занимает должность генерального директора Stranger Studios и является ведущим разработчиком Paid Memberships Pro — платформы подписок для WordPress. Уже более пяти лет он использует WordPress в разработке приложений на PHP. Джейсон рад помочь своим клиентам в получении оплаты через Paid Memberships Pro, позволяя им открывать новые и развивать существующие предприятия. Вы можете связаться с ним на сайте **therealjasoncoleman.com** или в "Твиттере", по псевдониму **@jason_coleman**.

Предметный указатель

—
_s, тема 143

1

landl, хостинг 416

A

Admin Columns, плагин 111
Adobe AIR, среда 25
Advanced Custom Fields, плагин 111, 347
Ajax
 ◇ вызов 37, 297
 ◇ запрос 289
 ◇ технология 293
 ◇ управление количеством запросов 302
Akismet, плагин 48, 89, 276
All In One WP Security & Firewall, плагин 277
Alternative PHP Cache, расширение 426
Amazon EC2, сервис 416
AMPPS, стек 86
Android Studio, среда 468
Android, ОС 25, 28
Angular, фреймворк 297
Apache Bench, утилита 402
 ◇ графическое представление результатов 405
 ◇ запуск 403
 ◇ тестирование с помощью 405
 ◇ установка 402
Apache, веб-сервер 37, 268, 412
 ◇ настройка 417
APCu, система 431
App Builder, приложение 474
AppCamera, плагин 479
AppPresser, инструмент 34, 473
 ◇ установка и конфигурация 473

AppWoo, плагин 483
AskApache Password Protect, плагин 278
Authorize.net, платежная система 214, 447
Automatic, компания 278, 360
AutoML Translation, функция 504

B

Backbone.js
 ◇ фреймворк 311
BackupBuddy, плагин 114, 276
Bad Behavior, плагин 276
BadgeOS Community, надстройка 99
BadgeOS, плагин 29, 99, 112
Base64, формат 322
Basecamp, приложение 28
Batcache, плагин 429
BatteryStatus, плагин 470
bbPress, плагин 29 42
Bootstrap
 ◇ импорт 145
Bootstrap Twitter, пользовательский интерфейс 144
Braintree Payments, платежная система 447
Browser Capabilities, проект 156
BuddyPress, плагин 29, 115, 336
 ◇ группы 42
 ◇ компоненты 122
 ◇ настройки 125
 ◇ плагины для 126
 ◇ поля профиля 125
 ◇ страницы 124
 ◇ таблицы базы данных 116

C

Calypso, приложение 360
Camera, плагин 470
Capture, плагин 470
Certbot, инструмент 269

Chrome, браузер 153

◊ панель отладки 399

Classic Editor, плагин 344

CLI Add-on, расширение 507

Configure SMTP, плагин 250

Connection, плагин 470

Constant Contact API 506

Cordova, веб-фреймворк 467

◊ iOS 470

◊ Android 468

◊ плагины 470

◊ установка 467

Cron, задание 243

◊ запуск с сервера 246

◊ использование серверного 247

CRUD-метод 316

CSRF-атака 320

CSS, язык описания 17, 25, 149, 472

CSV, формат 111, 115, 405

D

DesktopServer, инструмент 86

Device, плагин 471

DigitalOcean, хостинг 268, 416

Directions, сервис 503

Distance Matrix, сервис 503

DNS-сервер 416

Docker, инструмент 87

Dropbox API 506

Dropbox, сервис 114, 449

Drupal, система управления контентом 37

Duplicator WordPress Migration Plugin,
плагин 507

Dynamic Dummy Image Generator,
инструмент 491

E

Easy Digital Downloads, плагин 440, 444

◊ примеры кода 445

eBay API 505

Esma International, ассоциация 292

ECMAScript, стандарт 292

◊ версии ES6, ES9, ESNext 293

ElasticPress by 10up, плагин 502

Elasticsearch, сервис 502

Elevation, API-интерфейс 503

Events, плагин 471

Evernote, сервис 449

Exploit Scanner, плагин 278

F

Facebook Connect, плагин 36

Facebook, приложение 26

Faker, библиотека 501

File, плагин 471

Firebase, проект 487

Flickr API 505

Flywheel, платформа 86

Foundation Zurb, пользовательский
интерфейс 144

FTP-сервер 114

G

GamiPress, плагин 112

GD, библиотека 490

Gecko, браузер 153

Genesis, фреймворк 144

Geocoder PHP, библиотека 495

Geocoding, API-интерфейс 503

Geolocation, плагин 471

GitHub, сервис 449

Globalization, плагин 471

GlotPress, инструмент 392

◊ использование 393

◊ создание сервера 393

Gmail, приложение 26

Gnuplot, утилита 405

Google

◊ Analytics, сервис 30

◊ APIs Explorer 505

◊ Apps, сервис 449

◊ Chrome, браузер 25

◊ Lighthouse, инструмент 29

◊ Maps, приложение 26

◊ Maps, сервис 503

◊ Maps, сервис для WordPress 504

◊ Translate, сервис 504

◊ Vision, плагин 503

Goutte, инструмент 501

gPano, плагин 504

GraphQL, язык запросов 513

Gravity Forms, плагин 114, 372

Gutenberg Map Block for Google Maps,
плагин 504

Gutenberg, редактор 37, 291, 342, 514

◊ советы по работе 354

GZIP-сжатие 415

Н

Heartbeat API 255, 304

- ◇ инициализация 304, 307
- ◇ использование 304
- ◇ клиентский JavaScript-код 305, 308
- ◇ серверный PHP-код 306, 309
- Heartbeat Control, плагин 255
- Hello Dolly, плагин 47, 89
- Hide the Admin Bar, плагин 213
- HTML, язык разметки гипертекста 17, 25, 472
- HTTP, протокол 315
- ◇ заголовок 317
- ◇ запрос 316
- ◇ ответ 316
- ◇ сообщение 316, 318
- HTTPS, протокол 268, 270
- ◇ решение проблем 272

I

Imagick, инструмент 490
Imagine, библиотека 491
InAppBrowser, плагин 471
InfiniteWP, плагин 360
Instagram Graph API 505
Ionic Framework, фреймворк 471
iOS, ОС 25, 28

J

JavaScript, язык программирования 17, 25, 150, 291, 356, 472

- ◇ библиотеки 295
- ◇ код 291
- ◇ размещение кода 296
- ◇ фреймворки 311

Joomla, система управления контентом 37
jQuery, библиотека 17, 151, 294
JSON, формат файла 36, 293, 315
JWT-токен 322

К

KHTML, браузер 153

L

LAMP, стек 86
Laravel, среда 17
LearnDash, плагин 486
Let's Encrypt, сервис 269
Limit Login Attempts, плагин 278

LinkedIn API 506
Local, инструмент 86

M

MailChimp API 506
MainWP, плагин 360
MAMP, стек 86
ManageWP, сервис 360
Mandrill, сервис 250
Mashable, компания 410
MaxMind GeoIP, сервис 495
Media, плагин 471
Member Network Sites, расширение 372
Memberlite, тема 44, 143

- ◇ дочерняя 144
- ◇ код 45

Members, плагин 11, 214
Memcached, система 427
Microsoft Office, пакет 449
Modernizr.js, библиотека 152
More Privacy Options, плагин 372
Mozilla

- ◇ OBI, проект 112
- ◇ браузер 153

Multisite

- ◇ Global Media, плагин 372
- ◇ Global Search, плагин 373
- ◇ Plugin Manager, плагин 372
- ◇ Robots.txt Manager, плагин 373
- ◇ Tools Add-on, расширение 507

MVC

- ◇ архитектура 38
- ◇ фреймворки 311

MySQL, система управления базами данных 17 48

N

Nginx, веб-сервер 412

- ◇ настройка 421

Node.js, платформа 37, 291, 356
Notification, плагин 471
NS Cloner

- ◇ Site Copier, плагин 373

O

OAuth

- ◇ аутентификация 323
- ◇ поток 323

OS X, ОС 153

Р

Pagely, хостинг 87 416
Paid Memberships Pro, плагин 29, 35, 41,
213, 338, 372, 440, 443
◇ адреса 461
◇ блокирование доступа к страницам 456
◇ выбор дополнительных настроек 454
◇ выбор настроек электронной почты 454
◇ выбор параметров оплаты 453
◇ задание уровней подписки 450
◇ изменение поведения 458
◇ конфигурация страниц 452
◇ определение способа оплаты 449
◇ установка и активация 450
Payment Card Industry (PCI), стандарт 447
PayPal, платежная система 214, 447
PDF-документы
◇ генерация 491
PhoneGap, версия Cordova 467
Photoshop, программа 489
PHP, язык программирования 17
phpDocumentor, инструмент 500
PHPUnit, система 500
PHP-библиотека 488
PMPPro
◇ Network, плагин 41
◇ Register Helper, плагин 41, 214
Posts 2 Posts (P2P), плагин 112
Prefork, модуль 417
Push-уведомление 487

Q

Query Monitor, плагин 424

R

Rackspace, хостинг 416
React, фреймворк 37, 291, 297, 312, 356
Redis, система 428
REST API 314, 329, 512
◇ аутентификация 320
◇ назначение 318
◇ плагины 334
REST, стиль 315
Retina, экран 150
Ruby on Rails, среда 17
Ruby, язык программирования 36

S

SaaS, модель доступа 449
Safari, браузер 153
Salesforce API Explorer 505
SchoolPress, веб-приложение 26, 41
◇ бизнес-модель 41
◇ исходный код 41
◇ сайт 41
SchoolPress, приложение 26
SendGrid, сервис 250
ServerPress, компания 86
Siege, утилита 409
Site Health, инструмент 401
SiteGround, хостинг 416
Snappy, библиотека 491
Splashscreen, плагин 471
SQL-запрос 17, 109
SSL, протокол 268
◇ включение 268
◇ решение проблем 273
Status Bar, плагин 471
Storage, плагин 471
Street View, API-интерфейс 504
Stripe
◇ Payment Gateway, расширение 441
◇ платежная система 214, 447
Symfony, среда 17

T

Theme My Login, плагин 213
Translate WordPress with GTranslate, плагин
504
Trello, приложение 26
Twilio, платформа 504
Twitter, приложение 26

U

Unix, система 246
URL-адрес 37
User Registration, расширение 372
User Role Editor, плагин 113

V

Vagrant, инструмент 87
Vanilla JavaScript, фреймворк 297
Varnish
◇ прокси-сервер 429

VaultPress, плагин 277
Vibration, плагин 471
VirtualBox, инструмент 87
Vue.js, фреймворк 297

W

W3 Total Cache, плагин 113, 410
WAMP, стек 86
WebAIM 154
WebKit, движок для браузера 154
Whoops, библиотека 502
Windows, ОС 271
WooCommerce
◊ плагин 29, 111, 335, 440
◊ расширения 441
WordCamps, конференции 15
WordFence, плагин 277
WordPress 15
◊ Ajax-вызов 297
◊ API-интерфейсы 502
◊ core 28
◊ Google Maps 504
◊ PHP/MySQL 31
◊ REST API 314
◊ REST API версии 2, 320
◊ Site Health, инструмент 401
◊ VIP, хостинг 87
◊ авторизация в 271
◊ анатомия приложения 40
◊ асинхронная обработка 310
◊ безопасность 31, 34, 260
◊ гибкость 31, 46
◊ действие в обход 438
◊ доля на рынке 515
◊ зачем использовать 29
◊ как фреймворк 37
◊ когда не следует использовать 35
◊ критика 32
◊ локализация приложений 382
◊ масштабирование 33, 394
◊ многосайтовая версия 41
◊ многосайтовые сети 357
◊ мобильные приложения 464
◊ модель-представление-контроллер, схема 38
◊ настройки 234
◊ "обезглавленные" версии 512
◊ обнаружение браузера 154
◊ определение локали 383
◊ оптимизация 394
◊ основы 46
◊ отправка писем 249
◊ плагины 29, 30, 34, 89, 512
◊ поддержка сайтов 17
◊ расширение 88
◊ редактор системы 342
◊ система управления контентом 37
◊ создание веб-приложений 25
◊ стоимость 32
◊ структура базы данных 48
◊ структура каталогов 46
◊ темы 40, 48, 128
◊ управление контентом 29
◊ управление пользователями 30
◊ усиление защиты 263
◊ хостинг 87
◊ цикл 100
◊ ядро 191, 345
Worker, модуль 417
WP All Import, плагин 115
WP Engine, хостинг 87, 416, 496
WP Google Maps, плагин 504
WP Migrate DB Pro, плагин 507
WP Multi Network, плагин 373
WP MVC, плагин 39
WP Single Sign-On, плагин 329
◊ настройка 330
WP Store Locator, плагин 504
WP-API Basic-Auth, плагин 322
WP-CLI, интерфейс командной строки 360
WYSIWYG, редактор 29

X

XAMPP, стек 86
XML, формат 115

Y

Yii, среда 17
YIKES, Inc., плагин 79
Yeast SEO, плагин 111, 113

Z

Zebra Image, инструмент 491
Zippykid, хостинг 416

А

- Автор 199
- Аггарвала, Аншу 316
- Адаптивный дизайн 44, 148
- Администратор 199
- Анатомия приложения WordPress 40
- Андерсон, Аарон 154
- Архив
 - ◇ извлечение файлов 499
 - ◇ упаковка файлов 497
- Атака методом прямого перебора 260
- Аутентификация 332
 - ◇ OAuth 323
 - ◇ базовая 321
 - ◇ с помощью куки-файлов 320
 - ◇ трехногая 323

Б

- База данных
 - ◇ WordPress 48
 - ◇ содержимое 508
- Безопасность 260
 - ◇ имя пользователя admin 261
 - ◇ код 278
 - ◇ обновление 261
 - ◇ одноразовые коды 284
 - ◇ основные меры 261
 - ◇ пароль 261
 - ◇ плагины 276
 - ◇ по умолчанию 270
- Блок
 - ◇ активация редактора 347
 - ◇ для разработки пользовательского опыта 347
 - ◇ заменты плагина 515
 - ◇ использование для контента и дизайна 344
 - ◇ использование для представления функциональности 344
 - ◇ категории 348
 - ◇ ограничение типа 349
 - ◇ пример 345
 - ◇ редактор 343
 - ◇ создание 345
 - ◇ сохранение данных 353
 - ◇ шаблон 351, 514
- Блокирование
 - ◇ доступа к определенной странице 456

- ◇ доступа к файлам 459
- ◇ страниц для пользователей без подписки 458
- ◇ страницы по URL-адресу 456
- ◇ части страницы в PHP-коде 457
- ◇ части страницы с помощью короткого кода 457
- Блэкберн, Джон 424
- Браузер
 - ◇ определение 154, 155, 157
- Буферизация вывода 274
- Бэнкс, Алекс 356, 513
- Бэнкс, Фил 360

В

- Валидация 280
- Веб-браузер 25
- Веб-приложение 15, 25
 - ◇ возможности устройства 27
 - ◇ задачи 26
 - ◇ интерактивные элементы 26
 - ◇ компиляция 477
 - ◇ логины 27
 - ◇ прогрессивное 28
 - ◇ работа в автономном режиме 27
 - ◇ связывание нескольких 27
 - ◇ создание 25
 - ◇ ссылки между страницами 478
 - ◇ тестирование 477
 - ◇ уровни пользователей 27
 - ◇ файл манифеста 28
 - ◇ функции 26
- Веб-разработчик 15
- Веб-сайт 25
- Веб-скрапинг 501
- Веб-страница 25
- Виджет 220 221
 - ◇ встраивание 227
 - ◇ добавление 221
 - ◇ определение области 225
 - ◇ панели инструментов 228
 - ◇ собственный 231
 - ◇ удаление 229
- Владелец ресурса 324
- Вложение 158
- Внешний край 395
- Возможности
 - ◇ устройства 27
- Выполнение запросов 104

Г

Генерация изображений 489
Геолокация 494
◊ поддержка в веб-хостингах 496
Геотаргетинг 494
Гиббс, Пол 116
Глобальные переменные 100
Горджес, Бун 116
Графический интерфейс администратора 52
Гурли, Дэвид 316

Д

Данные
◊ из внешних API-интерфейсов 509
◊ из файлов 509
◊ руководство по привязке 510
Джейкоби, Джон 116
Дополнение 253

З

Заголовок 250
◊ добавление в плагины и темы 254
◊ добавление в файлы 253
Загруженные файлы 48
Загрузчик шаблонов 40
Запрос 326

И

Иерархия шаблонов 130
Инструменты для разработки 500
Интерактивные элементы 26
Интервал
◊ добавление 245
Интернационализация 382
Источник 395

К

Кастомные инструкции SQL 280
Кастомные таблицы 436
Кэш oEmbed 159
Класс-оболочка 188
◊ пользовательский 183
◊ создание 186
Клиент 324
Ключи метаданных 64
Комментарий 68

◊ добавление 70
◊ идентификатор 69
◊ метаданные 73
◊ обновление 70
◊ список 69
◊ удаление 70
◊ удаление метаданных 74
Конечная точка 326
◊ для проверки учетных данных пользователя 333
◊ добавление 329
Контроллер 38, 40
Корневой каталог 46
Коулман, Джейсон 35, 143
Коулман, Кимберли 143
Крокфорд, Дуглас 356
Куки-файлы 320
Кэширование
◊ базы данных 414
◊ выборочное 430
◊ объектное 414
◊ страничное 411
◊ фрагментарное 431

Л

Лема, Крис 16
Логины 27
Локализация 382
◊ необходимость 382
◊ организация файлов 390
◊ процесс 383
Локальная среда разработки 86

М

Малленвег, Мэтт 90 (убрать эту строку)
Маршрут 326
◊ добавление 329
◊ регистрация 329
Масштабирование 394
◊ в WordPress 394
Матсон, Джефф 255
Медиазапрос 149
Меню 146
◊ динамические 148
◊ навигационные 147
Мессенленер, Брайан 34, 517
Метаблок
◊ использование 182
Метаданные 179

Метаключ 56

Миграция 506

- ◊ куда переносятся данные 508
- ◊ между платформами 508
- ◊ откуда переносятся данные 508
- ◊ плагины 507
- ◊ сайтов 507
- ◊ сервера 507

Минимизация 413

Многосайтовая сеть

- ◊ администрирование 363
- ◊ настройка 360, 365
- ◊ обновления 366
- ◊ отображение доменов 370
- ◊ панель администратора 363
- ◊ плагины 365
- ◊ пользователи 364
- ◊ сайты 363
- ◊ структура базы данных 367
- ◊ темы 365

Многосайтовый режим

- ◊ альтернативы 359
- ◊ использование 357, 358
- ◊ основная функциональность 373
- ◊ плагины 371
- ◊ транзиенты 434

Мобильное приложение 27

- ◊ гибридное 466
- ◊ нативное 465
- ◊ с использованием WordPress 464
- ◊ сценарии использования 464

Модель 38, 39

Модель-представление-контроллер,
схема 38

Модуль 253

Модульное тестирование 500

Мулленберг, Мэтт 90, 342

- ◊ презентация 32

Мультиарендность 360

Н

Набор изменений 159

Настройки 234

- ◊ игнорирование стандартов 237
- ◊ использование стандартов 236
- ◊ страница 234

О

Обнаружение функций 151

Обновление 261

Общий регламент по защите данных 455

Одноразовые коды 284

Операционная система (ОС) 28

Определение

- ◊ дисплея 149
 - ◊ размера экрана и окна 151
 - ◊ устройства 149, 150, 153
- Оптимизация 394
- ◊ MySQL 422
 - ◊ в WordPress 394
 - ◊ запросов к базе данных 424

Ответ 328

Отказ в обслуживании 260

П

Пароль

- ◊ надежный 261 262
- ◊ плохой 262

Перевод

- ◊ создание и загрузка файлов 389
- ◊ сочетание с экранированием 389

Перезапись 238

- ◊ добавление правил 238
- ◊ сброс правил 239
- ◊ функции 240

Петерсон, Кларисса 148

Пилигрим, Марк 152

Питлинг, Энди 115

Плагин 30, 39, 47, 213

- ◊ бесплатный 111
 - ◊ брандмауэр/сканер 277
 - ◊ для блокировки спама 276
 - ◊ для защиты авторизации и пароля 278
 - ◊ для резервного копирования 276
 - ◊ дополнения к 99
 - ◊ пиратский 90
 - ◊ пользовательский 43 44
 - ◊ создание 91
 - ◊ сообщества 115
 - ◊ структура файла 92
 - ◊ установка 90
- Платежная система 447
- Повторно используемый блок 160
- Повышение производительности 434
- Подписчик 199

Поисковая оптимизация (SEO) 30, 113

Пользователь

- ◇ вставка новых метаданных 56
- ◇ добавление 194
- ◇ идентификатор 56
- ◇ изменение роли в зависимости от уровня подписки 460
- ◇ настройка таблицы 211
- ◇ обновление метаданных 56
- ◇ получение данных 192
- ◇ проверка полномочий 278
- ◇ роли 42 199
- ◇ создание нового 52
- ◇ сохранение данных 55
- ◇ удаление 54
- ◇ удаление метаданных 57
- ◇ уровни участия 42

Пользовательская

- ◇ таблица 102
- ◇ таксономия 43

Пользовательские типы записей 29, 42, 158

- ◇ использование 175
- ◇ определение 160
- ◇ регистрация 160

Пользовательский

- ◇ CSS 159
- ◇ запрос 160
- ◇ интерфейс 26

Порселло, Ева 356, 513

Пост 59

- ◇ идентификатор 64
- ◇ кастомные типы 359
- ◇ категории 75
- ◇ комментарии 68
- ◇ метаданные 169, 353
- ◇ обновление метаданных 65
- ◇ ограничение типа 350
- ◇ простой 183
- ◇ сохранение данных 64

Постоянная ссылка 47

Представление 38, 39

Премиум плагин 114

Прикладной программный интерфейс 314

Приложение 25

- ◇ обертка 473

Прогрессивное веб-приложение 28

- ◇ плагин 28
- ◇ поддержка 28

Пространство имен 326

Протокол передачи гипертекста 268

Профиль

- ◇ добавление полей 206

Процессно-ориентированный веб-сервер 421

Публикация 158

Р

Работа в автономном режиме 27

Размещение кода

- ◇ при разработке плагинов 129
- ◇ при разработке приложений 128
- ◇ при разработке тем 129

Разрешение 150

Расширение класса 204

Редактор 199

- ◇ блоков 342

Редакция 159

Редди, Сайлу 316

Резервное копирование 114, 275

Роли

- ◇ изменение в зависимости от уровня подписки 460
- ◇ по умолчанию 200
- ◇ пользователей 42
- ◇ проверка 200
- ◇ создание 202

С

Санация 280

Сервер 324

- ◇ развертывание 416

Сервис-воркер 28

Сеть доставки контента (CDN) 395, 415

Сжатие и архивация данных 496

- ◇ РНР-библиотеки 500

Сил, Клифф 360

Символическая ссылка 271

Сканирование 275

Скрапинг веб-страниц 510

Событие 83, 84, 243

- ◇ планирование единичного 246

Сообщение

- ◇ получение метаданных 64
- ◇ список 62
- ◇ удаление 62

Среда

- ◇ интегрирования 88
- ◇ производственная 88
- ◇ разработки 86 88

Стандартная общественная лицензия 35, 90

Страница 158

Страничное кэширование

◊ настройки 411

Строка

◊ подготовка с помощью функций перевода 386

Структура каталогов WordPress 46

Суперадминистратор 199

Сэйер, Марджори 316

Т

Таксономия 168

◊ использование 175

◊ создание пользовательской 171

Таунс, Фредерик 410

Текстовый домен 384

◊ настройка 384

Тема 39 48 128

◊ _s 143

◊ CPT 142

◊ Memberlite 143

◊ архива 175

◊ создание версий CSS-файлов 140

◊ фреймворки для разработки 142

◊ функции для работы 136

Термин 75

◊ идентификатор 79

◊ метаданные 79

◊ обновление 78

◊ удаление 78

Тестирование 396

Типы записей 158

Торговый счет 448

Тотти, Брайан 316

Транзиент 430

◊ API для работы 431

◊ в многосайтовом режиме 434

У

Управление

◊ контентом 29

◊ пользователями 30

Уровни участия пользователей 42

Участник 199

Ф

Фильтр 83 85

Флэнаган, Дэвид 356

Форма 26 114

Функции 49, 52, 60, 64, 69, 73, 76

Х

Хо, Рэй 116

Хостинг 86, 415

◊ выбор 87

◊ для WordPress-сайтов 416

Хуки 83, 134, 198, 441

◊ активации 203

Ц

Цикл WordPress 100

Ш

Шаблоны

◊ блок 514

◊ иерархия 130

◊ копирование 134

◊ страниц 131

◊ темы 136

◊ файл 175

Шорткоды 136, 216

◊ атрибуты 217

◊ вложенные 218

◊ полезные функции 220

◊ удаление 219

Э

Экранирование 104, 280, 389

Электронная коммерция 440

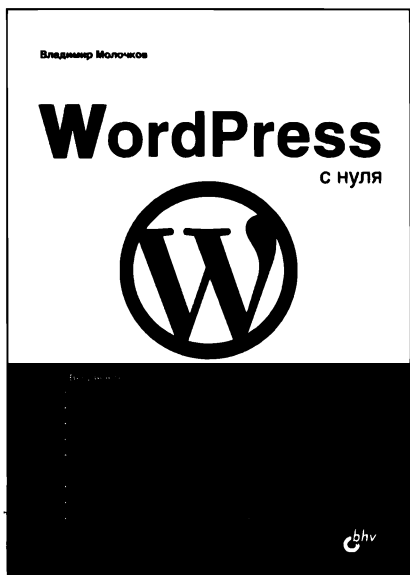
◊ плагины 440

Элемент меню навигации 159

Отдел оптовых поставок

E-mail: opt@bhv.ru

Создай свой сайт с WordPress!



- Как самостоятельно установить и настроить WordPress
- Как развернуть WordPress на локальном компьютере
- Как создавать современные веб-сайты
- Как подключать шаблоны оформления и разрабатывать собственные
- Как создать веб-сайт с адаптивным дизайном
- Какие плагины необходимы для вашего сайта
- Как запустить собственный интернет-магазин на WordPress
- Как выбрать хостинг для размещения веб-сайта или интернет-магазина
- Как перенести сайт в Интернет с локальной машины

Программа WordPress — бесплатная система управления содержимым сайта с открытым исходным кодом. С помощью WordPress можно создавать как простые веб-сайты «визитки», так и сложные проекты — блоги, корпоративные порталы, интернет-магазины.

Прочитав книгу, вы узнаете, как установить и настроить WordPress на локальном компьютере, как выбрать подходящий хостинг и перенести разработанный вами сайт в Интернет. Вы научитесь создавать сайты, адаптированные для компьютеров, ноутбуков и различных типов мобильных устройств. Книга поможет выбрать подходящий шаблон оформления или разработать собственный. Используя плагины, вы сможете превратить ваш сайт в популярный блог, корпоративный портал или интернет-магазин.

Эта книга поможет вам построить свой первый веб-сайт на WordPress без знаний программирования и навыков дизайна, даже если до этого вы никогда не занимались веб-разработкой.

Владимир Петрович Молочков, кандидат педагогических наук, преподаватель высшей квалификационной категории Политехнического колледжа Новгородского госуниверситета им. Ярослава Мудрого. Автор 18 книг по компьютерной тематике и более 200 статей в СМИ.

Разработка веб-приложений на WordPress

WordPress — это гораздо больше, чем платформа для создания блогов. Если у вас есть базовый опыт работы с PHP, HTML, CSS и JavaScript, вы можете использовать WordPress для создания быстрых, масштабируемых, безопасных и настраиваемых веб-приложений, мобильных приложений, веб-сервисов и целых сетей веб-сайтов. Наряду с основными функциями WordPress и взаимодействием с базой данных вы узнаете, как разрабатывать собственные плагины, темы и сервисы практически для любого вида веб-приложений или мобильных приложений.

В обновленном втором издании рассказывается о новых функциях и возможностях, добавленных в WordPress. Все примеры кода из книги доступны на веб-сервисе GitHub.

- Сравните WordPress с традиционными средами разработки приложений
- Используйте темы для настройки внешнего вида и плагины для обеспечения функциональности бэкенда
- Получите советы по выбору и созданию плагинов WordPress
- Регистрируйте собственные типы записей (CPT) и таксономии
- Управляйте учетными записями, ролями пользователей и доступом к данным
- Настраивайте асинхронное поведение с помощью jQuery
- Используйте WordPress для разработки мобильных приложений для iOS и Android
- Интегрируйте библиотеки PHP, внешние API и плагины веб-сервисов
- Получайте платежи с помощью плагинов eCommerce и membership
- Узнайте, как ускорить и масштабировать приложение WordPress
- Расширьте WordPress REST API и создайте собственные конечные точки (custom endpoints)
- Узнайте о разработке блоков WordPress Gutenberg

«WordPress – это не просто программное обеспечение, это движение, которое де-факто становится операционной системой Интернета. Когда вы научитесь использовать WordPress в качестве платформы для приложений, вы окажетесь на пике третьей волны его роста».

— Мэтт Малленвер,
соучредитель WordPress

Брайан Мессенленер — соучредитель нескольких веб-компаний, специализирующихся на разработке пользовательских и мобильных приложений на базе WordPress. Создавал решения на базе WordPress для таких клиентов, как журнал TIME, компании NBC, Microsoft и Uber.

Джейсон Коулман помог запустить несколько стартапов, используя WordPress в качестве фреймворка для приложений. В настоящее время возглавляет разработку Paid Memberships Pro — коммерческого плагина для организации платной подписки.

