Решения и примеры для программистов на PHP

PHP
Сборник рецептов

O'REILLY®

Дэвид Скляр и Адам Трахтенберг

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-059-6, название «PHP. Сборник рецептов» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

PHP Cookbook

David Sklar and Adam Trachtenberg

O'REILLY®

PHP

Сборник рецептов

Дэвид Скляр и Адам Трахтенберг

Символ

Санкт-Петербург — Москва

2005

Дэвид Скляр и Адам Трахтенберг

PHP. Сборник рецептов

Перевод А. Петухова

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научные редакторы	*Д. Горяинов, Р. Шевченко*
Редактор	*В. Овчинников*
Корректор	*С. Доничкина*
Верстка	*Н. Гриценко*

Скляр Д., Трахтенберг А.
PHP. Сборник рецептов. – Пер. с англ. – СПб: Символ-Плюс, 2005. – 672 с., ил.
ISBN 5-93286-059-6

«PHP. Сборник рецептов» Дэвида Скляра и Адама Трахтенберга содержит практические примеры и решения разнообразных задач, ежедневно возникающих перед программистами. Каждая задача снабжена проработанным решением – «рецептом», содержащим небольшой фрагмент кода, который можно вставлять прямо в приложение. Представлено более 250 рецептов – от самых простых, таких как посылка запроса в базу данных и получение доступа к URL, до полноценных программ, демонстрирующих более трудные задачи, например вывод HTML-таблиц и создание диаграмм. Рассмотрена работа со строками, числами, датами и временем, а также с массивами, файлами и каталогами. Обсуждаются переменные, функции, классы и объекты, регулярные выражения, шифрование и безопасность, интернет-службы, графика, интернационализация и локализация, PEAR, PHP в командной строке и PHP-GTK, формы, XML и доступ к базам данных.

Книга будет полезна всем, кто программирует на PHP, независимо от уровня их подготовки – от новичков до опытных профессионалов.

ISBN 5-93286-059-6
ISBN 1-56592-681-1 (англ)

© Издательство Символ-Плюс, 2005
Authorized translation of the English edition © 2002 O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 27.12.2004. Формат 70×100 1/16. Печать офсетная.
Объем 42 печ. л. Тираж 2000 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие . . . 15

1. Строки . . . 23

1.0. Введение . . . 23
1.1. Доступ к подстрокам . . . 26
1.2. Замещение подстрок . . . 27
1.3. Посимвольная обработка строк . . . 28
1.4. Пословный или посимвольный переворот строки . . . 30
1.5. Расширение и сжатие табуляций . . . 30
1.6. Управление регистром . . . 32
1.7. Включение функций и выражений в строки . . . 34
1.8. Удаление пробельных символов из строки . . . 35
1.9. Анализ данных, разделенных запятой . . . 36
1.10. Анализ данных, состоящих из полей фиксированной ширины . . . 37
1.11. Разбиение строк . . . 40
1.12. Упаковка текста в строки определенной длины . . . 42
1.13. Хранение двоичных данных в строках . . . 44

2. Числа . . . 47

2.0. Введение . . . 47
2.1. Проверка правильности записи числа в строке . . . 48
2.2. Сравнение чисел с плавающей точкой . . . 49
2.3. Округление чисел с плавающей точкой . . . 50
2.4. Работа с последовательностью целых чисел . . . 51
2.5. Генерация случайных чисел в пределах диапазона . . . 52
2.6. Генерация случайных чисел со смещением . . . 54
2.7. Взятие логарифмов . . . 55
2.8. Вычисление степеней . . . 56
2.9. Форматирование чисел . . . 57
2.10. Правильная печать слов во множественном числе . . . 57
2.11. Вычисление тригонометрических функций . . . 59
2.12. Тригонометрические вычисления не в радианах, а в градусах . . . 60
2.13. Работа с очень большими и очень маленькими числами . . . 61

2.14. Преобразование из одной системы счисления в другую 62
2.15. Вычисления с недесятичными числами 63

3. Дата и время 65

3.0. Введение 65
3.1. Определение текущей даты и времени 67
3.2. Преобразование времени и частей времени в метку времени UNIX 70
3.3. Преобразование метки времени в части времени и даты 71
3.4. Вывод на печать даты и времени в определенном формате 72
3.5. Определение разности между двумя датами 77
3.6. Определение разности между датами юлианского календаря 79
3.7. Определение дня недели, месяца, года или номера недели в году 81
3.8. Проверка корректности даты 82
3.9. Выделение дат и времен из строк 84
3.10. Сложение и вычитание дат 87
3.11. Учет часовых поясов при определении времени 88
3.12. Учет перехода на летнее время 93
3.13. Выработка высокоточного времени 94
3.14. Получение интервалов времени 95
3.15. Работа с негригорианскими календарями 96
3.16. Программа: Календарь 98

4. Массивы 101

4.0. Введение 101
4.1. Определение массива с ненулевым начальным индексом 104
4.2. Хранение множества элементов массива с одним ключом 105
4.3. Инициализация массива диапазоном целых чисел 106
4.4. Перебор элементов массива 107
4.5. Удаление элементов из массива 110
4.6. Изменение длины массива 112
4.7. Добавление одного массива к другому 114
4.8. Преобразование массива в строку 116
4.9. Печать массивов с запятыми 118
4.10. Проверка наличия ключа в массиве 119
4.11. Проверка наличия элемента в массиве 119
4.12. Определение позиции элемента в массиве 121
4.13. Нахождение элементов, удовлетворяющих определенному критерию 122
4.14. Нахождение элемента массива с наибольшим или наименьшим значением 123

4.15. Обращение массива . . . 124
4.16. Сортировка массива . . . 125
4.17. Сортировка массива по вычисляемому полю . . . 126
4.18. Сортировка множества массивов . . . 129
4.19. Сортировка массива с использованием метода вместо функции . . . 130
4.20. Рандомизация массива . . . 131
4.21. Тасование колоды карт . . . 132
4.22. Удаление двойных элементов из массива . . . 133
4.23. Определение объединения, пересечения или разности двух массивов . . . 134
4.24. Определение всех комбинаций элементов массива . . . 136
4.25. Нахождение всех перестановок массива . . . 139
4.26. Программа: Печать массива в виде HTML-таблицы . . . 141

5. Переменные . . . 145
5.0. Введение . . . 145
5.1. Операторы == и =: как избежать путаницы . . . 146
5.2. Установка значения по умолчанию . . . 147
5.3. Обмен значениями без временных переменных . . . 148
5.4. Создание динамического имени переменной . . . 149
5.5. Статические переменные . . . 150
5.6. Совместное использование переменных процессами . . . 152
5.7. Сериализация данных сложных типов в виде строки . . . 154
5.8. Получение дампа содержимого переменных в виде строк . . . 156

6. Функции . . . 160
6.0. Введение . . . 160
6.1. Доступ к параметрам функций . . . 161
6.2. Установка значений по умолчанию для параметров функции . . . 162
6.3. Передача значений по ссылке . . . 164
6.4. Именованные параметры . . . 165
6.5. Создание функции, принимающей переменное количество аргументов . . . 167
6.6. Возвращение значений по ссылке . . . 169
6.7. Возвращение более одного значения . . . 170
6.8. Пропуск определенных возвращаемых значений . . . 171
6.9. Возвращение информации об ошибке . . . 173
6.10. Вызов переменных функций . . . 174
6.11. Доступ к глобальной переменной внутри функции . . . 175
6.12. Создание динамических функций . . . 177

7. Классы и объекты . . . 178
7.0. Введение . . . 178
7.1. Реализация объектов . . . 182
7.2. Определение конструкторов объектов . . . 183
7.3. Уничтожение объекта . . . 184
7.4. Клонирование объектов . . . 185
7.5. Присваивание ссылок на объекты . . . 185
7.6. Применение методов к объекту, возвращенному другим методом . . . 186
7.7. Доступ к переопределенным методам . . . 187
7.8. Перегрузка свойств . . . 189
7.9. Полиморфизм методов . . . 190
7.10. Обнаружение методов и свойств объекта . . . 192
7.11. Добавление свойств в базовый объект . . . 194
7.12. Динамическое создание класса . . . 195
7.13. Динамическая реализация объекта . . . 196

8. Основы Web . . . 198
8.0. Введение . . . 198
8.1. Установка cookies . . . 200
8.2. Чтение значений cookie . . . 201
8.3. Удаление cookies . . . 202
8.4. Перенаправление по другому адресу . . . 203
8.5. Отслеживание сеанса работы с сайтом . . . 204
8.6. Хранение сеансов в базе данных . . . 205
8.7. Идентификация различных броузеров . . . 209
8.8. Формирование строки запроса GET . . . 211
8.9. Применение базовой аутентификации HTTP . . . 213
8.10. Аутентификация, основанная на cookies . . . 216
8.11. Передача выходной информации в броузер . . . 218
8.12. Буферизация вывода в броузер . . . 219
8.13. Сжатие веб-вывода с помощью gzip . . . 220
8.14. Сокрытие от пользователей сообщений об ошибках . . . 221
8.15. Настройка обработки ошибок . . . 222
8.16. Применение пользовательского обработчика ошибок . . . 225
8.17. Регистрация ошибок . . . 226
8.18. Устранение ошибок «headers already sent» (заголовки уже посланы) . . . 227
8.19. Регистрация отладочной информации . . . 229
8.20. Чтение переменных окружения . . . 231
8.21. Установка переменных окружения . . . 232

8.22. Чтение конфигурационных переменных . . . 233
8.23. Установка конфигурационных переменных . . . 235
8.24. Взаимодействие в рамках Apache . . . 235
8.25. Профилирование программы . . . 237
8.26. Программа: (Де)активатор учетной записи на веб-сайте . . . 240
8.27. Программа: Контролер злоумышленных пользователей . . . 242

9. Формы . . . 249

9.0. Введение . . . 249
9.1. Обработка информации, полученной из формы . . . 251
9.2. Проверка корректности введенных в форму данных . . . 253
9.3. Работа с многостраничными формами . . . 255
9.4. Повторный вывод форм с информацией и сообщениями об ошибках . . . 258
9.5. Защита от многократной отправки одной и той же формы . . . 261
9.6. Обработка загруженных файлов . . . 263
9.7. Организация безопасности обработки форм в PHP . . . 265
9.8. Пользовательские данные и escape-последовательности . . . 267
9.9. Обработка внешних переменных с точками в именах . . . 268
9.10. Использование элементов формы с несколькими вариантами значений . . . 270
9.11. Создание выпадающих меню на основе текущей даты . . . 271

10. Доступ к базам данных . . . 273

10.0. Введение . . . 273
10.1. Работа с базами данных, состоящих из текстовых файлов . . . 279
10.2. Работа с базами данных DBM . . . 280
10.3. Соединение с базой данных SQL . . . 284
10.4. Выполнение запросов к базе данных SQL . . . 286
10.5. Извлечение строк без цикла . . . 288
10.6. Модификация данных в базе данных SQL . . . 291
10.7. Эффективное повторение запросов . . . 292
10.8. Определение количества строк, возвращенных запросом . . . 294
10.9. Преобразование кавычек в escape-последовательности . . . 295
10.10. Регистрация отладочной информации и ошибок . . . 297
10.11. Автоматическое присваивание уникальных значений идентификаторов . . . 300
10.12. Программное создание запросов . . . 301
10.13. Постраничный вывод большого количества записей . . . 305
10.14. Кэширование запросов и результатов . . . 310
10.15. Программа: Хранение сообщений форума, разбитых на темы . . . 312

11. Автоматизация работы с Web 320

11.0. Введение 320
11.1. Получение содержимого URL методом GET 322
11.2. Извлечение содержимого URL с помощью метода POST 324
11.3. Получение содержимого URL, если требуется отправить cookies 326
11.4. Получение содержимого URL, требующее отправки заголовков 328
11.5. Получение содержимого HTTPS URL 329
11.6. Отладка обмена заголовками HTTP 330
11.7. Выделение информации на веб-странице 333
11.8. Извлечение ссылок из HTML-файла 334
11.9. Преобразование ASCII в HTML 336
11.10. Преобразование HTML в ASCII 337
11.11. Удаление тегов HTML и PHP 338
11.12. Использование шаблонов системы Smarty 339
11.13. Анализ файла протокола веб-сервера 341
11.14. Программа: обнаружение устаревших сылок 343
11.15. Программа: Обнаружение свежих ссылок 345

12. XML 349

12.0. Введение 349
12.1. Генерация XML вручную 352
12.2. Генерация XML с применением DOM 354
12.3. Анализ XML с помощью DOM 357
12.4. Анализ XML с помощью SAX 360
12.5. Преобразование XML с помощью XSLT 366
12.6. Посылка запросов XML-RPC 369
12.7. Прием запросов XML-RPC 372
12.8. Посылка SOAP-запросов 376
12.9. Прием SOAP-запросов 379
12.10. Обмен данными с помощью WDDX 382
12.11. Чтение RSS-рассылок 384

13. Регулярные выражения 387

13.0. Введение 387
13.1. Переход от ereg к preg 391
13.2. Поиск слов 393
13.3. Нахождение n-го совпадения 394
13.4. Выбор между поглощающим и непоглощающим сравнением 395
13.5. Проверка правильности адресов электронной почты 398

13.6. Поиск в файле всех строк, соответствующих шаблону . . . 401
13.7. Сборка текста, заключенного в теги HTML . . . 402
13.8. Экранирование специальных символов внутри регулярного выражения . . . 404
13.9. Чтение записей с шаблоном-разделителем . . . 405

14. Шифрование и безопасность . . . 407

14.0. Введение . . . 407
14.1. Не храните пароли на своем сайте . . . 409
14.2. Сокрытие данных при помощи кодирования . . . 410
14.3. Проверка данных с помощью хеширования . . . 410
14.4. Хранение паролей . . . 412
14.5. Проверка надежности пароля . . . 414
14.6. Работа с потерянными паролями . . . 416
14.7. Шифрование и дешифрование данных . . . 417
14.8. Хранение зашифрованных данных в файле или базе данных . . . 423
14.9. Совместное использование зашифрованных данных с другим веб-сайтом . . . 426
14.10. Обнаружение SSL-соединения . . . 428
14.11. Шифрование сообщений электронной почты с помощью GPG . . . 429

15. Графика . . . 432

15.0. Введение . . . 432
15.1. Рисование линий, прямоугольников и многоугольников . . . 436
15.2. Рисование дуг, эллипсов и окружностей . . . 438
15.3. Рисование узорными линиями . . . 440
15.4. Рисование текста . . . 441
15.5. Рисование центрированного текста . . . 444
15.6. Построение динамических изображений . . . 449
15.7. Создание и установка прозрачного цвета . . . 451
15.8. Безопасная работа с изображениями . . . 452
15.9. Программа: создание гистограмм результатов голосования . . . 454

16. Интернационализация и локализация . . . 458

16.0. Введение . . . 458
16.1. Перечень допустимых локалей . . . 460
16.2. Использование определенной локали . . . 460
16.3. Установка локали по умолчанию . . . 461
16.4. Локализация текстовых сообщений . . . 462
16.5. Локализация дат и времени . . . 466
16.6. Локализация денежных значений . . . 467

16.7. Локализация изображений . . . 470
16.8. Локализация включаемых файлов . . . 471
16.9. Управление ресурсами локализации . . . 472
16.10. Расширение gettext . . . 474
16.11. Чтение и запись символов Unicode . . . 475

17. Интернет-службы . . . 477

17.0. Введение . . . 477
17.1. Отправка почты . . . 478
17.2. Отправка почты в кодировке MIME . . . 481
17.3. Чтение почты с помощью IMAP или POP3 . . . 483
17.4. Отправка сообщений в новостные группы Usenet . . . 486
17.5. Чтение новостей из Usenet . . . 489
17.6. Получение и размещение файлов с помощью FTP . . . 494
17.7. Поиск адресов с помощью LDAP . . . 497
17.8. Применение LDAP для аутентификации пользователей . . . 499
17.9. Поиск в DNS . . . 502
17.10. Проверка функционирования хоста . . . 503
17.11. Получение информации о доменном имени . . . 505

18. Файлы . . . 507

18.0. Введение . . . 507
18.1. Создание или открытие локального файла . . . 511
18.2. Создание временного файла . . . 513
18.3. Открытие удаленного файла . . . 514
18.4. Чтение из стандартного потока ввода . . . 515
18.5. Чтение файла в строку . . . 515
18.6. Подсчет строк, абзацев или записей в файле . . . 517
18.7. Обработка каждого слова в файле . . . 519
18.8. Чтение определенной строки в файле . . . 521
18.9. Обработка файла по строкам или абзацам в обратном направлении . . . 522
18.10. Выбор случайной строки из файла . . . 522
18.11. Рандомизация всех строк в файле . . . 523
18.12. Обработка текстовых полей переменной длины . . . 524
18.13. Чтение файлов конфигурации . . . 525
18.14. Чтение или запись в определенное место в файле . . . 528
18.15. Удаление из файла последней строки . . . 529
18.16. Непосредственная модификация файла без временной копии . . . 531
18.17. Сброс вывода в файл . . . 532
18.18. Запись в стандартный поток вывода . . . 533

18.19. Запись в несколько файловых дескрипторов одновременно 534
18.20. Преобразование метасимволов среды в escape-последовательности 535
18.21. Передача входной информации в программу 537
18.22. Чтение из стандартного потока вывода программы 537
18.23. Чтение из стандартного потока ошибок программы 539
18.24. Блокировка файла 540
18.25. Чтение и запись сжатых файлов 543
18.26. Программа: Unzip 545

19. Каталоги 547

19.0. Введение 547
19.1. Получение и установка меток даты/времени файла 550
19.2. Получение информации о файле 551
19.3. Изменение прав доступа к файлу или его владельца 553
19.4. Разделение имени файла на составляющие 554
19.5. Удаление файла 556
19.6. Копирование и перемещение файла 556
19.7. Обработка всех файлов в каталоге 557
19.8. Получение списка имен файлов, соответствующих шаблону 558
19.9. Обработка всех файлов в каталоге 559
19.10. Создание новых каталогов 561
19.11. Удаление каталога и его содержимого 563
19.12. Программа: Перечень каталогов веб-сервера 564
19.13. Программа: Поиск сайта 568

20. PHP на стороне клиента 572

20.0. Введение 572
20.1. Анализ аргументов программы 577
20.2. Анализ аргументов программы с помощью getopt 578
20.3. Чтение ввода с клавиатуры 582
20.4. Чтение паролей 583
20.5. Показ в окне графических элементов управления 586
20.6. Показ в окне нескольких графических элементов управления 587
20.7. Реакция на действия пользователя 590
20.8. Показ меню 592
20.9. Программа: Командная оболочка 595
20.10. Программа: Служба погоды 598

21. PEAR 607
21.0. Введение 607
21.1. Работа с менеджером пакетов PEAR 610
21.2. Нахождение пакетов PEAR 612
21.3. Поиск информации о пакете 613
21.4. Установка пакетов PEAR 615
21.5. Установка пакетов PECL 616
21.6. Обновление пакетов PEAR 618
21.7. Удаление пакетов PEAR 619
21.8. Документирование классов с помощью PHPDoc 620

Алфавитный указатель 623

Предисловие

PHP лежит в основе миллионов динамических веб-приложений. Широкий набор возможностей, понятный синтаксис и поддержка различных операционных систем и веб-серверов сделали его идеальным языком как для быстрой разработки несложных веб-сайтов, так и для кропотливой и методичной работы над сложными веб-системами.

Одной из главных причин успеха PHP как языка веб-сценариев стало то, что он изначально задумывался как инструмент обработки HTML-форм и создания веб-страниц. Это сразу сделало PHP весьма дружественной средой разработки для Всемирной паутины. Кроме того, PHP – полиглот. PHP может общаться с большим количеством баз данных и знает многочисленные протоколы Интернета. PHP упрощает анализ данных броузера и может выполнять HTTP-запросы. Эта веб-специфическая направленность распространяется на все рецепты и примеры книги «PHP. Сборник рецептов».

Эта книга представляет собой сборник решений наиболее распространенных задач из практики PHP. Мы постарались включить материал, который будет привлекателен для всех – от новичков до мастеров. И если мы достигли цели, то вы узнаете что-нибудь новое (а возможно, многому научитесь). Здесь найдутся подсказки для тех, кто ежедневно программирует на PHP, и тех, кто пришел в PHP с опытом работы на других языках.

PHP – свободно распространяемая среда программирования. Его можно взять в виде исходного или двоичного кода с сайта *http://www.php.net/*. Веб-сайт PHP также содержит инструкции по установке, полную документацию и ссылки на источники в Интернете, рабочие группы пользователей, списки почтовой рассылки и другие ресурсы PHP.

Для кого эта книга

Это книга для программистов, решающих практические задачи с помощью PHP. Если же вы совсем не знаете PHP, то пусть это будет ваша вторая книга по PHP. Первой должна быть «Programming PHP», также от издательства O'Reilly & Associates.[1]

Тем, кто уже знаком с PHP, эта книга поможет преодолеть определенные трудности и будет верным спутником на всю жизнь (по крайней

[1] Lerdorf R., Tatroe K. «Programming PHP», O'Reilly, 2002.

мере, пока вы занимаетесь программированием). Кроме того, здесь показано, как решать конкретные задачи, такие как отправка почты или создание SOAP-сервера, способ решения которых на других языках, возможно, вам уже известен. Программисты, переводящие приложения с других языков на PHP, найдут в этой книге надежного помощника.

Что в этой книге

Мы не ожидаем, что вы сядете и прочитаете эту книгу от корки до корки (хотя будем счастливы, если вы это сделаете!). Программисты на PHP постоянно сталкиваются лицом к лицу с множеством сложных задач по широкому кругу вопросов. Обратитесь к этому сборнику, чтобы найти решение встретившейся задачи. Каждый рецепт представляет собой разъяснение, а это хорошая отправная точка на пути к успешному решению. Если в рецепте что-то не рассматривается подробно, то приводятся ссылки на соответствующие рецепты книги и другие онлайновые и автономные источники.

Если вы соберетесь прочитать всю главу сразу, это будет правильное решение. В основном сложность рецептов возрастает от начала к концу главы. В конце многих глав есть примеры программ, объединяющих рассмотренные примеры. Введение в начале каждой главы дает вводный обзор и основополагающие сведения о материале, рассматриваемом в этой главе, а также акцентирует внимание на особо важных рецептах.

Книга начинается с четырех глав, повествующих об основных типах данных. В главе 1 подробно рассматриваются такие вопросы, как обработка подстрок, управление регистром, разделение строки на более мелкие части и анализ данных, разделенных запятыми. Глава 2 посвящена операциям над числами с плавающей точкой, случайным числам, преобразованию чисел из одной системы счисления в другую и форматированию чисел. В главе 3 показано, как работать с датами и временем, как их форматировать, как работать с часовыми поясами и летним временем и как определять время с точностью до микросекунды. Глава 4 посвящена операциям с массивами, таким как выполнение циклов, объединение, обращение, сортировка и извлечение определенных элементов.

В следующих трех главах обсуждаются блоки, образующие программу. В главе 5 рассматриваются примечательные возможности PHP в области работы с переменными, такие как значения по умолчанию, статические переменные и создание строкового представления сложных типов данных. Рецепты в главе 6 имеют дело с применением функций в PHP: обработка аргументов, передача и возвращение переменных по ссылке, создание функций во время выполнения и использование области видимости переменных. Рецепты главы 7 посвящены объектно-ориентированным возможностям PHP, таким как использование пе-

резагрузки и полиморфизма, определение конструкторов и клонирование объектов.

Ядро этой книги составляют пять глав, посвященных центральным темам веб-программирования. Глава 8 рассматривает работу с cookies, заголовки, механизм аутентификации, переменные конфигурации и другие традиционные аспекты веб-приложений. В главе 9 идет речь об обработке и проверке подлинности ввода в формах, отображении форм с сообщениями об ошибках и о преобразовании в escape-последовательности специальных символов в пользовательских данных. В главе 10 объяснены различия между текстовым файлом, DBM и базами данных SQL, и на основе уровня абстракции базы данных PEAR DB показано, как присваивать уникальные значения идентификаторов, извлекать строки, изменять данные, преобразовывать кавычки в escape-последовательности и регистрировать отладочную информацию. В главе 11 основное внимание уделено извлечению URL и обработке HTML, но рассказано и о применении шаблонов и об анализе журнала доступа к серверу. Глава 12 посвящена XML и связанным с ним форматам, в том числе DOM, SAX, XSLT, XML-RPC и SOAP.

Следующий раздел книги представляет серию глав о других возможностях и расширениях PHP, которые составляют значительную часть полезной функциональности. Это рецепты, помогающие создавать приложения более устойчивые, защищенные, дружественные к пользователю и более эффективные. В главе 13 рассматриваются регулярные выражения, включая определение допустимых адресов электронной почты, выделение текста внутри тегов HTML и использование поглощающего и непоглощающего сравнения. В главе 14 обсуждается шифрование, включая генерирование и сохранение паролей, совместное использование закодированной информации, запись шифрованных данных в файл или базу данных и применение SSL. Глава 15 показывает, как динамически создавать графику с помощью рецептов по рисованию текста, линий, многоугольников и кривых. В главе 16 рассказано, как сделать приложения дружественными в мировом масштабе, и приведены рецепты по использованию локализации и локализованного текста, местных дат и времени, значений валюты и изображений. В главе 17 обсуждаются связанные с сетью Интернет задачи, такие как чтение и отправка электронной почты и сообщений новостных групп, использование FTP и LDAP, поиск с применением сервисов DNS и Whois.

Главы 18 и 19 посвящены файловой системе. Глава 18 фокусируется на файлах: их открытие и закрытие с использованием временных файлов, блокировка файла, посылка сжатых файлов и обработка содержимого файлов. Глава 19 имеет дело с каталогами и метаданными файлов. Она содержит рецепты по изменению прав доступа к файлу и прав владения, перемещению и удалению файла, обработке всех файлов каталога.

И наконец, две главы, расширяющие сферу деятельности PHP. Глава 20 посвящена использованию PHP вне веб-программирования. В ее рецептах рассматриваются темы, связанные с работой в командной строке, анализ аргументов программы и чтение паролей, темы, относящиеся к построению клиентских GUI-приложений с помощью PHP-GTK, например для отображения элементов управления окном, реагирования на действия пользователя и показа меню. В главе 21 рассказано о PEAR, расширении PHP и хранилище приложений. PEAR – это набор программ PHP, предоставляющий различные функции и расширения PHP. Мы используем модули PEAR на всем протяжении книги и показываем, как инсталлировать и обновлять их.

Другие источники

Веб-сайты

Существует огромное количество онлайновых справочных материалов по PHP. Быстрое интернет-соединение, предоставляющее все от аннотированных руководств по PHP до сайтов с периодическими статьями и учебными пособиями, достойно конкурирует с большой книжной полкой, заставленной книгами по PHP. Вот некоторые ключевые сайты:

Аннотированные сведения по PHP: http://www.php.net/manual/

Сайт переведен на семнадцать языков и включает как официальную документацию, так и возможности языка и комментарии пользователей.

Списки почтовых рассылок PHP: http://www.php.net/mailing-lists.php

Здесь много списков почтовых рассылок, посвященных инсталляции, программированию, расширению PHP и различным другим темам. Веб-интерфейс исключительно для чтения списков почтовых рассылок можно найти на *http://news.php.net/*.

Архивы презентаций PHP: http://conf.php.net/

Собрание презентаций PHP, показанных на различных конференциях.

PEAR: http://pear.php.net/

Здесь PEAR определяется как «структура и система распространения повторно используемых компонентов PHP». На этом сайте можно найти массу полезных PHP-классов и простых программ.

PHP.net: Путеводитель туриста: http://www.php.net/sites.php

Это руководство по различным веб-сайтам под эгидой *php.net*.

База знаний по PHP: http://php.faqts.com/

Множество вопросов и ответов от сообщества PHP, а также ссылки на другие источники.

PHP DevCenter: http://www.onlamp.com/php/

Коллекция статей и учебных пособий по PHP с хорошим сочетанием начальных и более сложных тем.

Книги

В этом разделе перечислены книги – полезные справочники и учебные пособия по созданию приложений с помощью PHP. Большинство из них посвящены веб-программированию; кроме того, понадобятся книги по MySQL, HTML, XML и HTTP.

В конце раздела мы перечислили несколько книг, полезных для любого программиста, независимо от выбранного языка. Эти работы могут улучшить ваши навыки программирования, научив думать о программировании как о части более масштабного процесса решения задачи.

- «Programming PHP» (Программирование на PHP) Кевина Тэтро (Kevin Tatroe) и Расмуса Лердорфа (Rasmus Lerdorf), O'Reilly.
- «HTML and XHTML: The Definitive Guide» Чака Муссиано (Chuck Musciano) и Билла Кеннеди (Bill Kennedy), O'Reilly.[1]
- «Dynamic HTML: The Definitive Guide» Дэнни Гудмена (Danny Goodman), O'Reilly.
- «Mastering Regular Expressions» Джеффри Фридла (Jeffrey E. F. Friedl), O'Reilly.[2]
- «XML in a Nutshell» Эллиотта Расти Харольда (Elliotte Rusty Harold) и У. Скотта Минса (W. Scott Means), O'Reilly.[3]
- «MySQL Reference Manual» Майкла «Монти» Видениуса (Michael «Monty» Widenius) и Дэвида Эксмарка (David Axmark), O'Reilly и MySQL AB (см. *http://www.mysql.com/documentation/*).
- «MySQL» Поля Дюбуа (Paul DuBois), New Riders.[4]
- «Web Security, Privacy, and Commerce» Симсона Гарфинкеля (Simson Garfinkel) и Жэне Спаффорда (Gene Spafford), O'Reilly.
- «Web Services Essentials» Этана Керами (Ethan Cerami), O'Reilly.
- «HTTP Pocket Reference» Клинтона Вонга (Clinton Wong), O'Reilly.
- «The Practice of Programming», Брайана У. Кернигана (Brian W. Kernighan) и Роба Пайка (Rob Pike), Addison-Wesley.

---

1 Чак Муссиано и Билл Кеннеди «HTML и XHTML. Подробное руководство», 4-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

2 Джеффри Фридл «Регулярные выражения», 2-е издание. – Пер. с англ. – СПб: Питер, 2003.

3 Эллиот Расти Харольд и Скотт Минс «XML. Справочник». – Пер. с англ. – СПб: Символ-Плюс, 2002.

4 Дюбуа П. «MySQL», 2-е издание. – Пер. с англ. – М : Вильямс, 2004.

- «Programming Pearls» Джона Луиса Бентли (Jon Louis Bentley), Addison-Wesley.[1]
- «The Mythical Man-Month» Фредерика П. Брукса (Frederick P. Brooks), Addison-Wesley.[2]

Соглашения, принятые в этой книге

Соглашения по программированию

В большинстве случаев мы опускали в примерах этой книги комбинации символов `<?php` и `?>` – открывающий и закрывающий теги, которые начинают и заканчивают программу, написанную на PHP, за исключением тех примеров, где тело программы включает открывающий или закрывающий тег. Для того чтобы свести к минимуму конфликты имен, названия функций и классов в данном сборнике начинаются с префикса `pc_`.

Примеры в этой книге были написаны для выполнения в версии PHP 4.2.2. Простые программы должны работать и в UNIX, и в Windows, за исключением случаев, особо упомянутых в тексте. Некоторые функции, особенно связанные с XML, были написаны для выполнения в версии PHP 4.3.0. Зависимость от возможностей, не представленных в версии PHP 4.2.2, отмечалась в тексте особо.

Типографские соглашения

В этой книге приняты следующие типографские соглашения:

Курсив

Для имен файлов и каталогов, адресов электронной почты и URL, а также для новых терминов в том месте, где они определяются.

`Моноширинный`

Для текстов программ и ключевых слов, имен переменных, функций, командных опций, параметров, классов и HTML-тегов.

`Моноширинный полужирный`

Для выделения строк вывода в листинге программы и командных строк, которые должен ввести пользователь.

`Моноширинный курсив`

Употребляется для обозначения элементов, которые надо заменить действительными значениями из вашей собственной программы.

---

[1] Бентли Дж. Л. «Жемчужины программирования», 2-е издание – Пер. с англ. – СПб: Питер, 2002.

[2] Фредерик Брукс «Мифический человеко-месяц». – Пер. с англ. – СПб: Символ-Плюс, 2000.

Комментарии и вопросы

Пожалуйста, направляйте комментарии и вопросы, касающиеся этой книги, издателю:

O'Reilly & Associates, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

(800) 998-9938 (в Соединенных Штатах или Канаде)

(707) 829-0515 (международный/местный)

(707) 829-0104 (факс)

Мы поддерживаем веб-страницу для этой книги, где приводим ошибки, примеры и другую дополнительную информацию. Вот ее адрес:

http://www.oreilly.com/catalog/phpckbk

Можно сделать замечание или задать вопрос, послав электронное письмо по адресу:

bookquestions@oreilly.com

За дополнительной информацией по этой книге, о конференциях, Resource Centers и O'Reilly Network обращайтесь к веб-сайту O'Reilly:

http://www.oreilly.com

Благодарности

Особая благодарность всем, кто пожертвовал своим временем, не пожалел творческих способностей и знаний, чтобы сделать PHP тем, что он в настоящее время собой представляет. Результатом этих поразительных усилий добровольцев стало создание не только сотни тысяч строк исходного кода, но и всеобъемлющей документации, инфраструктуры тестирования, множества дополнительных приложений и библиотек и бурно растущего сообщества пользователей во всем мире. Мы взволнованы представившейся нам почетной возможностью познакомить сообщество PHP с книгой «PHP. Сборник рецептов».

Благодарим также наших рецензентов: Стига Баккена (Stig Bakken), Шейна Каравео (Shane Caraveo), Айка де Лоренцо (Ike DeLorenzo), Расмуса Лэрдофа (Rasmus Lerdorf), Адама Мортона (Adam Morton), Офира Прусака (Ophir Prusak), Кевина Тэтроу (Kevin Tatroe) и Натана Торкингтона (Nathan Torkington). Они выловили значительное количество ошибок и внесли массу полезных предложений для улучшения этой книги. Мы бы хотели особенно отметить Ната Торкингтона (Nat Torkington), в изобилии снабдившего нас полезными изменениями и добавлениями.

Student.Net Publishing, Student.Com и TVGrid.Com обеспечили богатое окружение для экспериментов с PHP. Появление этой книги стало

возможным во многом благодаря нашему опыту работы в этих сообществах. Брет Мартин (Bret Martin) и Miranda Productions предоставили хостинг и другие средства, обеспечившие возможность нашей удаленной совместной работы во время написания книги. Мы находимся на расстоянии четырех миль друг от друга, но для Манхэттена это удаленно.

Последние по порядку, но не по значению, благодарности нашему редактору, Пауле Фергюсон (Paula Ferguson). Она твердой рукой провела «PHP. Сборник рецептов» через весь издательский процесс в O'Reilly – от того момента, когда она потрясающе быстро (с точки зрения наших друзей) приняла наше скромное предложение издать эту книгу, до окончательной обработки изменений, предложенных нами в последнюю минуту. Без нее эта книга никогда бы не превратилась из идеи в реальность.

Дэвид Скляр

Благодарю Адама за совместное написание этой книги (и вылавливание всех мест, где я поставил слишком много круглых скобок).

Спасибо моим родителям, которые не догадывались о том, к чему это приведет, когда 20 лет назад купили мне тот 4-килобайтный компьютер Radio Shack Color.

Благодарю Сюзанну (Susannah) за стойкую любовь и поддержку и за то, что в трудные минуты напоминала мне, что жизнь – это не параграф.

Адам Трахтенберг

Трудно выразить, как я обязан Дэвиду за совместную работу над книгой «PHP. Сборник рецептов». Его замечания радикально улучшили мое умение писать, а его непоколебимая пунктуальность помогала мне придерживаться графика работы.

Благодарю Coleco и их компьютер Adam за то, что они позволили мне вообразить себя первым ребенком, который владеет компьютером, названным в его честь.

Спасибо всем моим друзьям и одноклассникам по бизнес-школе, которым надоело слушать, как я говорю: «Извините, я сегодня вечером должен идти работать над книгой», и кто продолжал разговаривать со мной, после того как я отвечал на их звонки с двухнедельным опозданием.

Особые благодарности Элизабет Хондл (Elizabeth Hondl). Ее по-детски искренняя заинтересованность веб-технологиями доказывает, что если задавать достаточно много вопросов, то из них вполне может получиться книга.

Спасибо моему брату, родителям и всей семье, от которых я получил так много. Их одобрение и любовь поддерживают меня.

1

Строки

1.0. Введение

Строки в PHP – это последовательность символов, такая как «We hold these truths to be self evident», или «Жил да был», или даже «111211211». При чтении из файла или выводе в броузер данные представляются в виде строк.

Отдельные символы можно считать элементами индексированного массива, как в C. Первый символ в строке имеет нулевой индекс. Например:

```
$neighbor = 'Hilda';
print $neighbor[3];
d
```

Однако строки в PHP отличаются от строк в C тем, что они могут быть бинарными (то есть могут содержать символ NULL) и могут произвольно менять свой размер, который ограничен лишь объемом доступной памяти.

Строки можно задавать тремя способами: в одинарных кавычках, в двойных кавычках и в формате встроенного документа (heredoc). При использовании одинарных кавычек следует экранировать только два специальных символа: обратную косую черту и саму одинарную кавычку:

```
print 'I have gone to the store.';
print 'I\'ve gone to the store.';
print 'Would you pay $1.75 for 8 ounces of tap water?';
print 'In double-quoted strings, newline is represented by \n';
I have gone to the store.
I've gone to the store.
Would you pay $1.75 for 8 ounces of tap water?
In double-quoted strings, newline is represented by \n
(В строках с двойными кавычками символ новой строки представлен \n)
```

Задавать строки в одинарных кавычках проще и быстрее, поскольку в этом случае PHP не проверяет наличие переменных или почти всех escape-последовательностей. Строки в двойных кавычках не распознают escape-код одинарной кавычки, но распознают вставленные в строку переменные и escape-последовательности, представленные в табл. 1.1.

Таблица 1.1. Escape-последовательности строки в двойных кавычках

Escape-последовательность	Символ
\n	Новая строка (ASCII 10)
\r	Возврат каретки (ASCII 13)
\t	Табуляция (ASCII 9)
\\	Обратная косая черта
\$	Знак доллара
\"	Двойные кавычки
\{	Левая фигурная скобка
\}	Правая фигурная скобка
\[	Левая скобка
\]	Правая скобка
от \0 до \777	Восьмеричное значение
от \x0 до \xFF	Шестнадцатеричное значение

Например:

```
print "I've gone to the store.";
print "The sauce cost \$10.25.";
$cost = '$10.25';
print "The sauce cost $cost.";
print "The sauce cost \$\061\060.\x32\x35.";
I've gone to the store.
The sauce cost $10.25.
The sauce cost $10.25.
The sauce cost $10.25.
```

Последняя строчка кода печатает цену соуса правильно, поскольку символ 1 имеет десятичный ASCII-код 49 и восьмеричный код 061. Символ 0 имеет десятичный ASCII-код 48 и восьмеричный код 060; 2 имеет десятичный ASCII-код 50 и шестнадцатеричный код 32; а 5 имеет десятичный ASCII-код 53 и шестнадцатеричный код 35.

Строки встроенного документа распознают все те вхождения и escape-коды, что и строки в двойных кавычках, но при этом они допускают использование двойных кавычек. Встроенный документ начинается с <<< и метки. Эта метка (без ограничивающих ее пробельных символов) с точкой с запятой в конце оператора (если это необходимо) заканчивает встроенный документ. Например:

```
print <<< END
It's funny when signs say things like:
   Original "Root" Beer
   "Free" Gift
   Shoes cleaned while "you" wait
or have other misquoted words.
END;
It's funny when signs say things like:
   Original "Root" Beer
   "Free" Gift
   Shoes cleaned while "you" wait
or have other misquoted words.
```

При использовании встроенного документа сохраняются новые строки, интервалы и кавычки. Метка конца строки чувствительна к регистру, и согласно принятому соглашению она записывается в верхнем регистре. Таким образом, следующее выражение правильное:

```
print <<< PARSLEY
It's easy to grow fresh:
Parsley
Chives
on your windowsill
PARSLEY;
```

Как и следующее:

```
print <<< DOGS
If you like pets, yell out:
DOGS AND CATS ARE GREAT!
DOGS;
```

Встроенный документ удобен при выводе на печать документа HTML с включенными в него переменными:

```
if ($remaining_cards > 0) {
    $url = '/deal.php';
    $text = 'Deal More Cards';
} else {
    $url = '/new-game.php';
    $text = 'Start a New Game';
}
print <<< HTML
There are <b>$remaining_cards</b> left.
<p>
<a href="$url">$text</a>
HTML;
```

Здесь точка с запятой после ограничителя конца строки необходима, поскольку она сообщает PHP о конце оператора. Однако в некоторых случаях точку с запятой ставить не надо:

```
$a = <<< END
Once upon a time, there was a
```

```
END
. ' boy!';
print $a;
Once upon a time, there was a boy!
```

Точки с запятой нет, поскольку данное выражение необходимо продолжить на следующей строке. Заметим также, что оператор конкатенации строк и ограничитель конца строки следует располагать в разных строках для того, чтобы PHP смог распознать ограничитель.

1.1. Доступ к подстрокам

Задача

Предположим, что требуется выделить часть строки, начиная с определенной позиции. Например, необходимы первые восемь символов имени пользователя, введенного в форму.

Решение

Для выделения подстроки применяется функция `substr()`:

```
$substring = substr($string,$start,$length);
$username = substr($_REQUEST['username'],0,8);
```

Обсуждение

Если `$start` и `$length` больше нуля, то функция `substr()` возвращает `$length` символов строки, начиная с позиции `$start`. Первый символ находится в нулевой позиции:

```
print substr('watch out for that tree',6,5);
out f
```

Если пропустить `$length`, то функция `substr()` возвратит строку, начиная с позиции `$start` до конца первоначальной строки:

```
print substr('watch out for that tree',17);
t tree
```

Если `$start` плюс `$length` указывает на позицию за концом, то функция `substr()` возвращает всю строку до конца, начиная с позиции `$start`:

```
print substr('watch out for that tree',20,5);
ree
```

Если число `$start` отрицательное, то функция `substr()` определяет начальную позицию подстроки, считая от конца строки к началу:

```
print substr('watch out for that tree',-6);
print substr('watch out for that tree',-17,5);
t tree
out f
```

Если число $length отрицательное, то substr() определяет конечную позицию подстроки, считая от конца строки к началу:

```
print substr('watch out for that tree',15,-2);
print substr('watch out for that tree',-4,-1);
hat tr
tre
```

См. также

Документацию по функции substr() на *http://www.php.net/substr*.

1.2. Замещение подстрок

Задача

Требуется заменить подстроку другой строкой. Например, перед тем как напечатать номер кредитной карты, вы хотите скрыть все цифры ее номера, за исключением последних четырех.

Решение

Используйте функцию substr_replace():

```
// Все, начиная с позиции $start до конца строки $old_string,
// заносится в $new_substring
$new_string = substr_replace($old_string,$new_substring,$start);

// $length символов, начиная с позиции $start, заменяются на $new_substring
$new_string = substr_replace($old_string,$new_substring,$start,$length);
```

Обсуждение

Без аргумента $length, функция substr_replace() заменяет все, начиная с позиции $start до конца строки. Если значение $length определено, то замещается только это количество:

```
print substr_replace('My pet is a blue dog.','fish.',12);
print substr_replace('My pet is a blue dog.','green',12,4);
$credit_card = '4111 1111 1111 1111';
print substr_replace($credit_card,'xxxx ',0,strlen($credit_card)-4);
My pet is a fish.
My pet is a green dog.
xxxx 1111
```

Если число $start отрицательное, то новая подстрока помещается в позицию $start, считая от конца строки $old_string, а не с начала:

```
print substr_replace('My pet is a blue dog.','fish.',-9);
print substr_replace('My pet is a blue dog.','green',-9,4);
My pet is a fish.
My pet is a green dog.
```

Если $start и $length равны 0, то новая подстрока вставляется в начало $old_string:

```
print substr_replace('My pet is a blue dog.','Title: ',0,0);
Title: My pet is a blue dog.
```

Функция substr_replace() удобна, когда текст невозможно отобразить за один раз, и вы хотите показать его часть со ссылкой на остальное содержание. Например, следующее выражение отображает 25 строк текста с многоточием в качестве ссылки на страницу с продолжением:

```
$r = mysql_query("SELECT id,message FROM messages WHERE id = $id") or die();
$ob = mysql_fetch_object($r);
printf('<a href="more-text.php?id=%d">%s</a>',
       $ob->id, substr_replace($ob->message,' ...',25));
```

На странице *more-text.php* для извлечения полного текста сообщения и его отображения может использоваться идентификатор сообщения, переданный в строке запроса.

См. также

Документацию по функции substr_replace() на *http://www.php.net/substr-replace*.

1.3. Посимвольная обработка строк

Задача

Нужно обработать каждый символ строки по отдельности.

Решение

Цикл по символам строки с помощью оператора for. В этом примере подсчитываются гласные в строке:

```
$string = "This weekend, I'm going shopping for a pet chicken.";
$vowels = 0;
for ($i = 0, $j = strlen($string); $i < $j; $i++) {
    if (strstr('aeiouAEIOU',$string[$i])) {
        $vowels++;
    }
}
```

Обсуждение

Посимвольная обработка – это самый простой способ подсчета последовательности «Смотри и говори»:

```
function lookandsay($s) {
    // инициализируем возвращаемое значение пустой строкой
    $r = '';
    // переменная $m, которая содержит подсчитываемые символы,
```

```
    // инициализируется первым символом * в строке
    $m = $s[0];
    // $n, количество обнаруженных символов $m, инициализируется значением 1
    $n = 1;
    for ($i = 1, $j = strlen($s); $i < $j; $i++) {
        // если символ совпадает с последним символом
        if ($s[$i] == $m) {
            // увеличиваем на единицу значение счетчика этих символов
            $n++;
        } else {
            // иначе добавляем значение счетчика и символа
               к возвращаемому значению //
            $r .= $n.$m;
            // устанавливаем искомый символ в значение текущего символа //
            $m = $s[$i];
            // и сбрасываем счетчик в 1 //
            $n = 1;
        }
    }
    // возвращаем построенную строку, а также последнее значение
       счетчика и символ //
    return $r.$n.$m;
}

for ($i = 0, $s = 1; $i < 10; $i++) {
    $s = lookandsay($s);
    print "$s\n";
}
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
13211311123113112211
```

Это называется последовательностью «Смотри и говори», поскольку каждый элемент мы получаем, глядя на предыдущие элементы и говоря, сколько их. Например, глядя на первый элемент, 1, мы говорим «один». Следовательно, второй элемент – «11», то есть две единицы, поэтому третий элемент – «21». Он представляет собой одну двойку и одну единицу, поэтому четвертый элемент – «1211» и т. д.

См. также

Документацию по оператору `for` на *http://www.php.net/for*; дополнительную информацию о последовательности «Смотри и говори» на *http://mathworld.wolfram.com/LookandSaySequence.html*.

1.4. Пословный или посимвольный переворот строки

Задача

Требуется перевернуть слова или символы в строке.

Решение

Для посимвольного переворота строки применяется функция `strrev()`:

```
print strrev('This is not a palindrome.');
.emordnilap a ton si sihT
```

Чтобы перевернуть строку пословно, надо разобрать строку на слова, перевернуть слова, а затем собрать их заново в строку:

```
$s = "Once upon a time there was a turtle.";
// разбиваем строку на слова
$words = explode(' ',$s);
// обращаем массив слов
$words = array_reverse($words);
// $s = join(' ',$words);
print $s;
turtle. a was there time a upon Once
```

Обсуждение

Пословное обращение строки может быть также выполнено в одной строке:

```
$reversed_s = join(' ',array_reverse(explode(' ',$s)));
```

См. также

Рецепт 18.7, в котором рассматриваются последствия применения непробельных символов в качестве границы слов; документацию по функции `strrev()` на *http://www.php.net/strrev* и по функции `array_reverse()` на *http://www.php.net/array-reverse*.

1.5. Расширение и сжатие табуляций

Задача

Нужно заменить пробелы на табуляцию (или табуляцию на пробелы) и в то же время сохранить выравнивание теста по позициям табуляции. Например, вы хотите отобразить для пользователя текст стандартным образом.

Решение

Для замены пробелов на табуляцию или табуляции на пробелы следует применять функцию `str_replace()`:

```
$r = mysql_query("SELECT message FROM messages WHERE id = 1") or die();
$ob = mysql_fetch_object($r);
$tabbed = str_replace(' ',"\t",$ob->message);
$spaced = str_replace("\t",' ',$ob->message);

print "With Tabs: <pre>$tabbed</pre>";
print "With Spaces: <pre>$spaced</pre>";
```

Однако если для преобразования применяется функция `str_replace()`, то позиции табуляции нарушаются. Если вы хотите ставить табуляцию через каждые восемь символов, то в строке, начинающейся с пятибуквенного слова и табуляции, необходимо заменить табуляцию на три пробела, а не на один. Для замены табуляции на пробелы с учетом позиций табуляции следует применять функцию `pc_tab_expand()`, показанную в примере 1.1.

Пример 1.1. pc_tab_expand()

```
function pc_tab_expand($a) {
  $tab_stop = 8;
  while (strstr($a,"\t")) {
    $a = preg_replace('/^([^\t]*)(\t+)/e',
                      "'\\1'.str_repeat(' ',strlen('\\2') *
                       $tab_stop - strlen('\\1') % $tab_stop)",$a);
  }
  return $a;
}

$spaced = pc_tab_expand($ob->message);
```

Для обратной замены пробелов на табуляцию можно воспользоваться функцией `pc_tab_unexpand()`, показанной в примере 1.2.

Пример 1.2. pc_tab_unexpand()

```
function pc_tab_unexpand($x) {
  $tab_stop = 8;

  $lines = explode("\n",$x);
  for ($i = 0, $j = count($lines); $i < $j; $i++) {
    $lines[$i] = pc_tab_expand($lines[$i]);
    $e = preg_split("/(.\{$tab_stop})/",$lines[$i],-
1,PREG_SPLIT_DELIM_CAPTURE);
    $lastbit = array_pop($e);
    if (!isset($lastbit)) { $lastbit = ''; }
    if ($lastbit == str_repeat(' ',$tab_stop)) { $lastbit = "\t"; }
    for ($m = 0, $n = count($e); $m < $n; $m++) {
      $e[$m] = preg_replace('/  +$',"\t",$e[$m]);
    }
```

```
    $lines[$i] = join(``,$e).$lastbit;
  }
  $x = join("\n", $lines);
  return $x;
}

$tabbed = pc_tab_unexpand($ob->message);
```

Обе функции принимают в качестве аргумента строку и возвращают ее, модифицировав соответствующим образом.

Обсуждение

Каждая функция предполагает наличие позиций табуляции через каждые восемь пробелов, но это можно изменить, задав переменную `$tab_stop`.

Регулярное выражение в `pc_tab_expand()` соответствует и группе табуляций, и всему тексту в строке перед группой табуляций. Оно должно соответствовать тексту перед табуляциями, поскольку от длины этого текста зависит количество пробелов, замещающих табуляции, а последующий текст должен быть выровнен по позиции следующей табуляции. Эта функция не просто заменяет каждую табуляцию на восемь пробелов; она выравнивает текст, стоящий после табуляции, по позициям табуляций.

Точно так же функция `pc_tab_unexpand()` не только ищет восемь последовательных пробелов, а затем заменяет их одним символом табуляции. Она делит каждую строку на участки по восемь символов, а затем замещает пробелы в конце этих участков (по крайней мере два пробела) на табуляции. Это не только сохраняет выравнивание текста по позициям табуляций, но и сохраняет пробелы в строке.

См. также

Документацию по функции `str_replace()` на *http://www.php.net/str-replace*.

1.6. Управление регистром

Задача

Необходимо переключиться на прописные буквы, или в нижний регистр или иным образом изменить регистр символов в строке. Например, требуется сделать прописными начальные буквы имен и сохранить нижний регистр остальных букв.

Решение

Первые буквы одного или более слов можно сделать прописными с помощью функции `ucfirst()` или функции `ucwords()`:

```
print ucfirst("how do you do today?");
print ucwords("the prince of wales");
How do you do today?
The Prince Of Wales
```

Регистр всей строки изменяется функцией `strtolower()` или функцией `strtoupper()`:

```
print strtoupper("i'm not yelling!");
// Стандарт XHTML требует, чтобы символы в тегах были в нижнем регистре
print strtolower('<A HREF="one.php">one</A>');
I'M NOT YELLING!
<a href="one.php">one</a>
```

Обсуждение

Первый символ строки можно сделать прописным посредством функции `ucfirst()`:

```
print ucfirst('monkey face');
print ucfirst('1 monkey face');
Monkey face
1 monkey face
```

Обратите внимание, что во второй строке вывода слово «monkey» начинается со строчной буквы.

Функция `ucwords()` позволяет сделать прописным первый символ каждого слова в строке:

```
print ucwords('1 monkey face');
print ucwords("don't play zone defense against the philadelphia 76-ers");
1 Monkey Face
Don't Play Zone Defense Against The Philadelphia 76-ers
```

Как и следовало ожидать, функция `ucwords()` не делает прописной букву «t» в слове «don't». Но она также не делает прописной букву «e» в «70-e». Для функции `ucwords()` слово – это любая последовательность непробельных символов, за которой расположен один или несколько пробельных. Символы «'» и «-» не являются пробельными, поэтому функция `ucwords()` не считает «t» в «don't» или «e» в «70-e» начальными символами слов.

Ни `ucfirst()`, ни `ucwords()` не изменяют регистр не первых символов:

```
print ucfirst('macWorld says I should get a iBook');
print ucwords('eTunaFish.com might buy itunaFish.Com!');
MacWorld says I should get a iBook
ETunaFish.com Might Buy ItunaFish.Com!
```

Функции `strtolower()` и `strtoupper()` работают с целыми строками, а не только с отдельными символами. Функция `strtolower()` переводит все алфавитные символы в нижний регистр, а функция `strtoupper()` – в верхний:

```
print strtolower("I programmed the WOPR and the TRS-80.");
print strtoupper('"since feeling is first" is a poem by e. e. cummings.');
i programmed the wopr and the trs-80.
"SINCE FEELING IS FIRST" IS A POEM BY E. E. CUMMINGS.
```

При определении верхнего и нижнего регистров приоритетными для этих функций являются их локальные настройки.[1]

См. также

Дополнительную информацию о локальных настройках в главе 16; документацию по функции `ucfirst()` на *http://www.php.net/ucfirst*, по функции `ucwords()` на *http://www.php.net/ucwords*, по функции `strtolower()` на *http://www.php.net/strtolower* и по функции `strtoupper()` на *http://www.php.net/strtoupper*.

1.7. Включение функций и выражений в строки

Задача

Вставить результаты выполнения функции или выражения в строку.

Решение

Когда значение, которое необходимо вставить в строку, не может быть в нее включено, следует применять оператор конкатенации строк (.):

```
print 'You have '.($_REQUEST['boys'] + $_REQUEST['girls']).' children.';
print "The word '$word' is ".strlen($word).' characters long.';
print 'You owe '.$amounts['payment'].' immediately';
print "My circle's diameter is ".$circle->getDiameter().' inches.';
```

Обсуждение

Можно поместить переменные, свойства объекта и элементы массива (если индекс не в кавычках) непосредственно в строку в двойных кавычках:

```
print "I have $children children.";
print "You owe $amounts[payment] immediately.";
print "My circle's diameter is $circle->diameter inches.";
```

Точно так же непосредственная вставка или конкатенация строк работает во встроенном документе. Вставка с помощью конкатенации строк во встроенном документе может выглядеть немного странно, поскольку ограничитель встроенного документа и оператор конкатенации должны располагаться в разных строках:

---

[1] Для того чтобы все эти функции работали с русскими буквами, необходимо установить в окружении сценария нужную локализацию с помощью функции `setlocale()`. – *Примеч. науч. ред.*

```
print <<< END
Right now, the time is
END
. strftime('%c') . <<< END
 but tomorrow it will be
END
. strftime('%c',time() + 86400);
```

Кроме того, если вы производите вставку во встроенный документ, не забудьте добавить пробелы так, чтобы вся строка выглядела правильно. В предыдущем примере строка «Right now the time» должна включать замыкающий пробел, а строка «but tomorrow it will be» должна включать пробелы в начале и в конце.

См. также

Описание синтаксиса размещения так называемых «переменных переменных» (таких как `${"amount_$i"}`) в рецепте 5.4; документацию по оператору конкатенации строк на *http://www.php.net/language.operators.string*.

1.8. Удаление пробельных символов из строки

Задача

Надо удалить пробельные символы в начале или в конце строки. Например, привести в порядок данные, введенные пользователем, прежде чем счесть их действительными.

Решение

Следует обратиться к функциям `ltrim()`, `rtrim()` или `trim()`. Функция `ltrim()` удаляет пробельные символы в начале строки, `rtrim()` – в конце строки, а функция `trim()` – и в начале, и в конце строки:

```
$zipcode = trim($_REQUEST['zipcode']);
$no_linefeed = rtrim($_REQUEST['text']);
$name = ltrim($_REQUEST['name']);
```

Обсуждение

Эти функции считают пробельными следующие символы: символ новой строки, возврат каретки, пробел, горизонтальную и вертикальную табуляции и символ NULL.

Удаление пробельных символов из строки позволяет сэкономить память и может сделать более корректным отображение форматированных данных или текста, например, содержащегося в тегах `<pre>`. При проверке пользовательского ввода сначала нужно обрезать пробелы, так чтобы не заставлять того, кто ввел «98052 » вместо своего почтово-

го индекса, исправлять ошибку, которая, собственно, таковой не является. Отбрасывание начальных и конечных пробелов в тексте перед его проверкой означает, например, что «salami\n» будет равно «salami». Неплохо также нормализовать данные путем обрезания пробелов перед их занесением в базу данных.

Кроме того, функция `trim()` способна удалять из строки символы, определенные пользователем. Удаляемые символы передаются в качестве второго аргумента. Можно указать интервал символов с помощью двоеточия между первым и последним символом интервала.

```
// Удаление цифр и пробела в начале строки
print ltrim('10 PRINT A$',' 0..9');
// Удаление точки с запятой в конце строки
print rtrim('SELECT * FROM turtles;',';');
PRINT A$
SELECT * FROM turtles
```

PHP рассматривает `chop()` как синоним `rtrim()`. Однако лучше использовать `rtrim()`, поскольку поведение функции `chop()` в PHP отличается от поведения `chop()` в Perl (вместо которой в любом случае лучше применять функцию `chomp()`), а ее применение может затруднить другим чтение вашего кода.

См. также

Документацию по функции `trim()` на *http://www.php.net/trim*, по функции `ltrim()` на *http://www.php.net/ltrim* и по функции `rtrim()` на *http://www.php.net/rtrim*.

1.9. Анализ данных, разделенных запятой

Задача

Есть данные, разделенные запятыми (формат CSV), например, файл, экспортированный из Excel или из базы данных, и необходимо извлечь записи и поля в формате, с которым можно работать в PHP.

Решение

Если CSV-данные представляют собой файл (или они доступны через URL), то откройте файл с помощью функции `fopen()` и прочитайте данные с помощью функции `fgetcsv()`. Данные будут представлены в виде HTML-таблицы:

```
$fp = fopen('sample2.csv','r') or die("can't open file");
print "<table>\n";
while($csv_line = fgetcsv($fp,1024)) {
    print '<tr>';
    for ($i = 0, $j = count($csv_line); $i < $j; $i++) {
        print '<td>'.$csv_line[$i].'</td>';
```

```
    }
    print "</tr>\n";
}
print '</table>\n';
fclose($fp) or die("can't close file");
```

Обсуждение

Второй аргумент в `fgetcsv()` должен превышать максимальную длину строки в вашем CSV-файле. (Не забудьте посчитать пробельные символы, ограничивающие строку.) Если длина читаемой строки превышает 1 Кбайт, то число 1024, использованное в данном рецепте, надо заменить на действительную длину строки.

Функции `fgetcsv()` можно передать необязательный третий параметр, ограничитель, используемый вместо запятой. Однако применение другого ограничителя до некоторой степени лишает смысла CSV как наиболее простого способа обмена табличными данными.

Не старайтесь избегать функции `fgetcsv()`, а просто читайте строку и вызывайте функцию `explode()` в случае запятых. CSV является более сложным, когда имеешь дело с внедренными запятыми и двойными кавычками. Использование функции `fgetcsv()` защитит ваш код от трудноуловимых ошибок.

См. также

Документацию по функции `fgetcsv()` на *http://www.php.net/fgetcsv*.

1.10. Анализ данных, состоящих из полей фиксированной ширины

Задача

Необходимо разбить на части записи фиксированной ширины в строке.

Решение

Это делается при помощи функции `substr()`:

```
$fp = fopen('fixed-width-records.txt','r') or die ("can't open file");
while ($s = fgets($fp,1024)) {
    $fields[1] = substr($s,0,10);       // первое поле:
                                           первые 10 символов строки
    $fields[2] = substr($s,10,5);       // второе поле:
                                           следующие 5 символов строки
    $fields[3] = substr($s,15,12);      // третье поле:
                                           следующие 12 символов строки
    // функция обработки полей
    process_fields($fields);
}
fclose($fp) or die("can't close file");
```

Или функции `unpack()`:

```
$fp = fopen('fixed-width-records.txt','r') or die ("can't open file");
while ($s = fgets($fp,1024)) {
    // ассоциативный массив с ключами "title", "author" и "publication_year"
    $fields = unpack('A25title/A14author/A4publication_year',$s);
    // функция обработки полей
    process_fields($fields);
}
fclose($fp) or die("can't close file");
```

Обсуждение

Данные, в которых каждому полю выделено фиксированное число символов в строке, могут выглядеть, как этот список книг, названий и дат опубликования:

```
$booklist=<<<END
Elmer Gantry             Sinclair Lewis1927
The Scarlatti InheritanceRobert Ludlum 1971
The Parsifal Mosaic      Robert Ludlum 1982
Sophie's Choice          William Styron1979
END;
```

В каждой строке название занимает 25 символов, имя автора – следующие 14 символов, а год публикации – следующие 4 символа. Зная ширину полей, очень просто с помощью функции `substr()` перенести поля в массив.

```
$books = explode("\n",$booklist);

for($i = 0, $j = count($books); $i < $j; $i++) {
  $book_array[$i]['title'] = substr($books[$i],0,25);
  $book_array[$i]['author'] = substr($books[$i],25,14);
  $book_array[$i]['publication_year'] = substr($books[$i],39,4);
}
```

Разбиение переменной `$booklist` на массив строк позволяет применить один код разбора одной строки ко всем строкам, прочитанным из файла.

Цикл можно сделать более гибким, определив отдельные массивы для имен полей и их ширины, которые могут быть переданы в анализирующую функцию, как показано в функции `pc_fixed_width_substr()` примера 1.3.

Пример 1.3. pc_fixed_width_substr( )

```
function pc_fixed_width_substr($fields,$data) {
  $r = array();
  for ($i = 0, $j = count($data); $i < $j; $i++) {
    $line_pos = 0;
    foreach($fields as $field_name => $field_length) {
      $r[$i][$field_name] = rtrim(substr($data[$i],$line_pos,$field_length));
      $line_pos += $field_length;
    }
```

```
  }
  Return $r;
}

$book_fields = array('title' => 25,
                     'author' => 14,
                     'publication_year' => 4);

$book_array = pc_fixed_width_substr($book_fields,$books);
```

Переменная `$line_pos` отслеживает начало каждого поля, и она увеличивается на ширину предыдущего поля по мере того, как код обрабатывает каждую строку. Для удаления пробельных символов в конце каждого поля предназначена функция `rtrim()`.

Как альтернатива функции `substr()` для извлечения полей может применяться функция `unpack()`. Вместо того чтобы задавать имена полей и их ширину в виде ассоциативных массивов, создайте строку форматирования для функции. Код для извлечения полей фиксированной ширины аналогичен функции `pc_fixed_width_unpack()`, показанной в примере 1.4.

Пример 1.4. pc_fixed_width_unpack( )

```
function pc_fixed_width_unpack($format_string,$data) {
  $r = array();
  for ($i = 0, $j = count($data); $i < $j; $i++) {
    $r[$i] = unpack($format_string,$data[$i]);
  }
  return $r;
}

$book_array = pc_fixed_width_unpack('A25title/A14author/A4publication_year',
                                    $books);
```

Формат `A` означает «строку в обрамлении пробелов», поэтому нет необходимости удалять завершающие пробелы с помощью функции `rtrim()`.

Поля, перенесенные с помощью какой-либо функции в переменную `$book_array`, могут быть отображены в виде HTML-таблицы, например:

```
$book_array = pc_fixed_width_unpack('A25title/A14author/A4publication_year',
                                    $books);
print "<table>\n";
// печатаем строку заголовка
print '<tr><td>';
print join('</td><td>',array_keys($book_array[0]));
print "</td></tr>\n";
// печатаем каждую строку данных
foreach ($book_array as $row) {
    print '<tr><td>';
    print join('</td><td>',array_values($row));
    print "</td></tr>\n";
}
print '</table>\n';
```

Объединение данных с помощью тегов `</td><td>` формирует строку таблицы, не включая в нее начальный `<td>` и заключительный `</td>` теги. Печатая `<tr><td>` перед выводом объединенных данных и `</td></tr>` вслед за выводом объединенных данных, мы формируем полную строку таблицы.

И функция `substr()`, и функция `unpack()` имеют одинаковые возможности, если поля фиксированной ширины содержат строки, но функция `unpack()` является наилучшим решением при наличии полей других типов данных.

См. также

Более подробную информацию о функции `unpack()` в рецепте 1.13 и по адресу *http://www.php.net/unpack*; рецепт 4.8, который посвящен функции `join()`.

1.11. Разбиение строк

Задача

Необходимо разделить строку на части. Например, нужно получить доступ к каждой из строк, которые пользователь вводит в поле `<textarea>` формы.

Решение

Если в качестве разделителя частей строк выступает строковая константа, то следует применять функцию `explode()`:

```
$words = explode(' ','My sentence is not very complicated');
```

Функция `split()` или функция `preg_split()` применяются, если при описании разделителя требуется регулярное выражение POSIX или Perl:

```
$words = split(' +','This sentence  has  some extra whitespace  in it.');
$words = preg_split('/\d\. /','my day: 1. get up 2. get dressed 3. eat toast');
$lines = preg_split('/[\n\r]+/',$_REQUEST['textarea']);
```

В случае чувствительного к регистру разделителя применяется функция `spliti()` или флаг `/i` в функции `preg_split()`:

```
$words = spliti(' x ','31 inches x 22 inches X 9 inches');
$words = preg_split('/ x /i','31 inches x 22 inches X 9 inches');
```

Обсуждение

Простейшим решением из всех приведенных выше является использование `explode()`. Передайте ей разделитель строки, саму строку, которую необходимо разделить, и в качестве необязательного параметра предельное количество возвращаемых элементов:

```
$dwarves = 'dopey,sleepy,happy,grumpy,sneezy,bashful,doc';
$dwarf_array = explode(',',$dwarves);
```

Теперь переменная `$dwarf_array` — это массив из семи элементов:

```
print_r($dwarf_array);
Array
(
    [0] => dopey
    [1] => sleepy
    [2] => happy
    [3] => grumpy
    [4] => sneezy
    [5] => bashful
    [6] => doc
)
```

Если заданный предел меньше количества возможных частей, то последняя часть содержит все остальное:

```
$dwarf_array = explode(',',$dwarves,5);
print_r($dwarf_array);
Array
(
    [0] => dopey
    [1] => sleepy
    [2] => happy
    [3] => grumpy
    [4] => sneezy,bashful,doc
)
```

Функция `explode()` трактует разделитель строки буквально. Если разделитель строки определяется как запятая с пробелом, то данная функция делит строку по пробелу, следующему за запятой, а не по запятой или пробелу.

Функция `split()` предоставляет большую гибкость. Вместо строкового литерала в качестве разделителя она использует регулярное выражение POSIX:

```
$more_dwarves = 'cheeky,fatso, wonder boy, chunky,growly, groggy, winky';
$more_dwarf_array = split(', ?',$more_dwarves);
```

Это регулярное выражение разделяет строку по запятой, за которой следует необязательный пробел, что позволяет правильно определить всех новых гномов. Таким образом, пробелы в их именах не разделяют их на части, но каждое имя выделяется независимо от того, отделяется ли оно с помощью запятой «,» или с помощью запятой с пробелом «, »:

```
print_r($more_dwarf_array);
Array
(
    [0] => cheeky
    [1] => fatso
    [2] => wonder boy
```

```
        [3] => chunky
        [4] => growly
        [5] => groggy
        [6] => winky
    )
```

Существует функция preg_split(), которая подобно функции split() использует Perl-совместимые регулярные выражения вместо регулярных выражений POSIX. Функция preg_split() предоставляет преимущества различных расширений регулярных выражений в Perl, а также хитрые приемы, такие как включение текста-разделителя в возвращаемый массив строк:

```
$math = "3 + 2 / 7 - 9";
$stack = preg_split('/ *([+\-\/*]) */',$math,-1,PREG_SPLIT_DELIM_CAPTURE);
print_r($stack);
Array
(
    [0] => 3
    [1] => +
    [2] => 2
    [3] => /
    [4] => 7
    [5] => -
    [6] => 9
)
```

Разделитель-регулярное выражение ищет математические операторы (+, -, /, *), окруженные необязательными начальными или завершающими пробелами. Флаг PREG_SPLIT_DELIM_CAPTURE приказывает функции preg_split() включить совпадения как часть разделителя-регулярного выражения, заключенного в кавычки, в возвращаемый строковый массив. В кавычках только символы математических операций, поэтому возвращенный массив не содержит пробелов.

См. также

Документацию по функции explode() на *http://www.php.net/explode*, по функции split() на *http://www.php.net/split* и по функции preg_split() на *http://www.php.net/preg-split*. Регулярные выражения более подробно рассматриваются далее.

1.12. Упаковка текста в строки определенной длины

Задача

Необходимо упаковать линии текста в строку. Например, нужно отобразить текст, содержащийся в тегах <pre>/</ pre>, в пределах окна броузера обычного размера.

Решение

Это делается при помощи функции `wordwrap()`:

```
$s = "Four score and seven years ago our fathers brought forth on this
continent a new nation, conceived in liberty and dedicated to the proposition
that all men are created equal.";

print "<pre>\n".wordwrap($s)."\n</pre>";
<pre>
Four score and seven years ago our fathers brought forth on this continent
a new nation, conceived in liberty and dedicated to the proposition that
all men are created equal.
</pre>
```

Обсуждение

По умолчанию функция `wordwrap()` упаковывает текст в строки по 75 символов. Необязательный второй аргумент позволяет изменять длину строки:

```
print wordwrap($s,50);
Four score and seven years ago our fathers brought
forth on this continent a new nation, conceived in
liberty and dedicated to the proposition that all
men are created equal.
```

Для указания конца строки можно использовать не только символы «\n». Для получения двойного интервала между строками используйте «\n\ n»:

```
print wordwrap($s,50,"\n\n");
Four score and seven years ago our fathers brought

forth on this continent a new nation, conceived in

liberty and dedicated to the proposition that all

men are created equal.
```

В функции есть необязательный четвертый аргумент, управляющий обработкой слов, длина которых превышает указанную длину строки. Если этот аргумент равен 1, то слова разбиваются на части. В противном случае они простираются за указанный предел длины строки:

```
print wordwrap('jabberwocky',5);
print wordwrap('jabberwocky',5,"\n",1);
jabberwocky

jabbe
rwock
y
```

См. также

Документацию по функции `wordwrap()` на *http://www.php.net/wordwrap*.

1.13. Хранение двоичных данных в строках

Задача

Необходимо проанализировать строку, которая содержит значения, закодированные с помощью двоичной структуры, или закодировать значения в строку. Например, нужно сохранить числа в их двоичном представлении, а не как последовательность ASCII-символов.

Решение

Для сохранения двоичных данных в строке применяется функция `pack()`:

```
$packed = pack('S4', 1974, 106, 28225, 32725);
```

Функция `unpack()` позволяет извлекать двоичные данные из строки:

```
$nums = unpack('S4', $packed);
```

Обсуждение

Первым аргументом функции `pack()` является строка формата, который описывает способ кодировки данных, передаваемых остальными аргументами. Строка формата `S4` указывает, что функция `pack()` должна сформировать из входных данных четыре беззнаковых коротких целых (short) 16-битных числа в соответствии с машинным порядком байтов. В качестве входа даны числа 1974, 106, 28225 и 32725, а возвращаются восемь байт: 182, 7, 106, 0, 65, 110, 213 и 127. Каждая двухбайтная пара соответствует входному числу: $7 \times 256 + 182 = 1974$; $0 \times 256 + 106 = 106$; $110 \times 256 + 65 = 28225$; $127 \times 256 + 213 = 32725$.

Первый аргумент функции `unpack()` – это тоже строка формата, а второй аргумент представляет декодируемые данные. В результате передачи строки формата, равной `S4`, восьмибайтная последовательность, произведенная функцией `pack()`, приводит к получению четырехэлементного массива исходных чисел:

```
print_r($nums);
Array
(
    [1] => 1974
    [2] => 106
    [3] => 28225
    [4] => 32725
)
```

В функции `unpack()` за символом форматирования и его множителем может следовать строка, которая выступает в качестве индекса массива. Например:

```
$nums = unpack('S4num', $packed);
print_r($nums);
```

```
Array
(
    [num1] => 1974
    [num2] => 106
    [num3] => 28225
    [num4] => 32725
)
```

В функции `unpack()` несколько символов форматирования должны разделяться символом /:

```
$nums = unpack('S1a/S1b/S1c/S1d',$packed);
print_r($nums);
Array
(
    [a] => 1974
    [b] => 106
    [c] => 28225
    [d] => 32725
)
```

В табл. 1.2 приведены символы форматирования, которые можно использовать в функциях `pack()` и `unpack()`.

Таблица 1.2. Символы форматирования функций pack( ) и unpack( )

Символ форматирования	Тип данных
a	Строка, дополненная символами NUL
A	Строка, дополненная пробелами
h	Шестнадцатеричная строка, первый полубайт младший
H	Шестнадцатеричная строка, первый полубайт старший
c	`signed char`
C	`unsigned char`
s	`signed short` (16 бит, машинный порядок байтов)
S	`unsigned short` (16 бит, машинный порядок байтов)
n	`unsigned short` (16 бит, обратный порядок байтов)
v	`unsigned short` (16 бит, прямой порядок байтов)
i	`signed int` (машинно-зависимый размер и порядок байтов)
I	`unsigned int` (машинно-зависимый размер и порядок байтов)
l	`signed long` (32 бита, машинный порядок байтов)
L	`unsigned long` (32 бита, машинный порядок байтов)
N	`unsigned long` (32 бита, обратный порядок байтов)
V	`unsigned long` (32 бита, прямой порядок байтов)
f	`float` (машинно-зависимый размер и представление)

Таблица 1.2 (продолжение)

Символ форматирования	Тип данных
d	double (машинно-зависимый размер и представление)
x	NUL-байт
X	Возврат на один байт
@	Заполнение нулями по абсолютному адресу

Для a, A, h и H число после символа форматирования означает длину строки. Например, A25 означает строку из 25 символов, дополненную пробелами. В случае других символов форматирования это число означает количество данных указанного типа, последовательно появляющихся в строке. Остальные возможные данные можно указать при помощи символа «*».

С помощью функции unpack() можно преобразовывать различные типы данных. Этот пример заполняет массив ASCII-кодами каждого символа, находящегося в $s:

```
$s = 'platypus';
$ascii = unpack('c*',$s);
print_r($ascii);
Array
(
    [1] => 112
    [2] => 108
    [3] => 97
    [4] => 116
    [5] => 121
    [6] => 112
    [7] => 117
    [8] => 115
)
```

См. также

Документацию по функции pack() на *http://www.php.net/pack* и по функции unpack() на *http://www.php.net/unpack*.

2

Числа

2.0. Введение

В повседневной жизни идентифицировать число не трудно. Это и текущее время (например, 15:00), и $1.29 (цена какого-нибудь товара). Это может быть число π, равное отношению длины окружности к ее диаметру. Числа могут быть достаточно большими, например число Авогадро, равное примерно 6×10^{23}. В PHP числа могут представлять все, что здесь было перечислено.

Однако PHP не трактует все эти числа как «числа», а делит их на две группы: целые и числа с плавающей точкой. Целые – это целые числа, такие как –4, 0, 5 и 1 975. Числа с плавающей точкой – это десятичные числа, такие как –1,23; 0,0; 3,14159 и 9,9999999999.

Удобно, что в основном PHP сам отслеживает различия между этими двумя типами данных и автоматически преобразует целые в числа с плавающей точкой, а числа с плавающей точкой – в целые. Это с легкостью позволяет игнорировать скрытые детали. Это также означает, что `3/2` равно `1.5`, а не `1`, как это было бы в некоторых языках программирования. PHP также автоматически конвертирует строки в числа и обратно. Например, `1+"1"` равно `2`.

Однако иногда это беззаботное игнорирование приводит к ошибкам. Во-первых, числа не могут быть бесконечно большими или бесконечно малыми; существует минимальное значение, равное 2.2e–308, и максимальное значение, приблизительно равное 1.8e308.[1] Если нужны числа большие (или меньшие), то необходимо использовать библиотеки BCMath или GMP, которые рассматриваются в рецепте 2.13.

Кроме того, числа с плавающей точкой не могут быть абсолютно точными, а только с точностью до некоторого малого значения. В настоя-

[1] На самом деле эти значения зависят от платформы, но они общеприняты, поскольку описаны в 64-битном стандарте 754 IEEE.

щее время это значение достаточно мало в большинстве случаев, но при определенных обстоятельствах можно столкнуться с проблемами. Например, люди автоматически преобразовывают число 6 с бесконечной последовательностью девяток после десятичной точки в число 7, но PHP думает, что это число 6 с группой девяток. Поэтому если запросить у PHP целое значение этого числа, то он возвратит 6, а не 7. По тем же причинам, если цифра в двухсотой десятичной позиции является значимой, то числа с плавающей точкой бесполезны. И снова на выручку приходят библиотеки BCMath и GMP. Но в большинстве случаев поведение PHP при работе с числами безупречно и позволяет обращаться с ними как в реальной жизни.

2.1. Проверка правильности записи числа в строке

Задача

Вы хотите быть уверенным, что строка содержит число. Например, требуется проверить правильность возраста, введенного пользователем в поле ввода формы.

Решение

Обратитесь к функции `is_numeric()`:

```
if (is_numeric('five')) { /* false */ }

if (is_numeric(5))      { /* true  */ }
if (is_numeric('5'))    { /* true  */ }

if (is_numeric(-5))     { /* true  */ }
if (is_numeric('-5'))   { /* true  */ }
```

Обсуждение

Помимо работы с числами функция `is_numeric()` может также применяться к числовым строкам. Разница здесь в том, что целое число `5` и строка `5` в PHP с технической точки зрения не идентичны.[1]

Конечно, полезно, что функция `is_numeric()` правильно анализирует десятичные числа, такие как `5.1`; однако числа с разделителем тысячного разряда, такие как `5,100`, заставляют функцию `is_numeric()` вернуть `false`.

Для того чтобы убрать разделитель тысячного разряда из числа, перед функцией `is_numeric()` вызывается функция `str_replace()`:

[1] Наиболее ярко это различие проявляется при переходе от PHP 3 к PHP 4. В PHP 3 `empty('0')` возвращала `false`, а в PHP 4 она возвращает `true`. С другой стороны, `empty(0)` всегда возвращала `true` и продолжает это делать. (В действительности для переменных, содержащих `'0'` и `0`, следует вызывать `empty()`.) Более подробную информацию можно найти во введении.

```
is_numeric(str_replace($number, ',', ''));
```

Для проверки числа на принадлежность к определенному типу существует множество связанных функций с именами, не требующими объяснений: `is_bool()`, `is_float()` (или `is_double()` или `is_real()`; все это одно и то же) и `is_int()` (или `is_integer()` или `is_long()`).

См. также

Документацию по функции `is_numeric()` на *http://www.php.net/is-numeric* и по функции `str_replace()` на *http://www.php.net/str-replace*.

2.2. Сравнение чисел с плавающей точкой

Задача

Необходимо проверить равенство двух чисел с плавающей точкой.

Решение

Задайте малую дельту и проверьте числа на равенство в пределах этой дельты:

```
$delta = 0.00001;

$a = 1.00000001;
$b = 1.00000000;

if (abs($a - $b) < $delta) { /* $a и $b равны */ }
```

Обсуждение

Числа с плавающей точкой представляются в двоичном виде только с конечным количеством разрядов для мантиссы и порядка. При превышении этого количества происходит переполнение. В результате иногда PHP (а также другие языки) не считают два числа действительно равными, так как они могут отличаться в самом последнем разряде.

Для того чтобы обойти эту трудность, вместо проверки равенства `$a == $b` следует обеспечить очень небольшую разность (`$delta`) между первым и вторым числом. Размер этой дельты должен быть меньше разницы между двумя числами, которую вы хотите обеспечить. Затем для получения абсолютного значения разности вызывается функция `abs()`.

См. также

Рецепт 2.3 для получения информации по округлению чисел с плавающей точкой; документацию по числам с плавающей точкой в PHP на *http://www.php.net/language.types.float*.

2.3. Округление чисел с плавающей точкой

Задача

Необходимо округлить число с плавающей точкой или до целого значения, или до некоторого количества десятичных знаков.

Решение

Для того чтобы округлить число до ближайшего целого, предназначена функция `round()`:

```
$number = round(2.4);   // $number = 2
```

Округление до ближайшего большего целого выполняется при помощи функции `ceil()`:

```
$number = ceil(2.4);    // $number = 3
```

Функция `floor()` позволяет округлить число до ближайшего меньшего целого:

```
$number = floor(2.4);   // $number = 2
```

Обсуждение

Если число находится точно между двумя целыми, то поведение функции не определено:

```
$number = round(2.5);   // $number is 2 or 3!
```

Будьте осторожны! Мы упоминали в рецепте 2.2, что числа с плавающей точкой не всегда выражаются точным значением, поскольку это зависит от способа их внутреннего представления в компьютере. Это может создать ситуацию, когда очевидного ответа не существует. Вместо ожидаемого «0,5» значение может быть «,499999...9» (вся группа состоит их девяток) или «,500000...1» (с многими нулями и завершающей единицей). Если вы хотите быть уверенным, что число округляется в большую сторону, добавьте небольшую дельту перед округлением:

```
$delta = 0.0000001;
$number = round(2.5 + $delta);   // $number = 3
```

Для получения нужного количества десятичных знаков после запятой функция принимает необязательный аргумент, задающий точность. Например, для определения общей стоимости покупок в тележке покупателя:

```
$cart = 54.23;
$tax = $cart * .05;
$total = $cart + $tax;       // $total = 56.9415

$final = round($total, 2);   // $final = 56.94
```

См. также

Рецепт 2.2 для получения информации по сравнению чисел с плавающей точкой; документацию по функции `round()` на *http://www.php.net/round*.

2.4. Работа с последовательностью целых чисел

Задача

Требуется применить некоторый код к диапазону целых чисел.

Решение

Это делается при помощи функции `range()`, которая возвращает массив, состоящий из целых чисел:

```
foreach(range($start,$end) as $i) {
    plot_point($i);
}
```

Иногда вместо функции `range()` целесообразно применить цикл `for`. Для инкремента можно использовать также значения, отличные от 1. Например:

```
for ($i = $start; $i <= $end; $i += $increment) {
    plot_point($i);
}
```

Обсуждение

Циклы, подобные приведенному выше, являются общепринятыми. Например, вы могли бы разрабатывать функцию и должны были бы вычислить результаты для массива точек на графике. Или вести обратный отсчет в NASA перед запуском космического челнока Колумбия.

В первом примере функция `range()` возвращает массив значений от `$start` до `$end`. Затем `foreach` берет каждый элемент и присваивает его переменной `$i` внутри цикла. Преимущество применения функции `range()` в ее краткости, но этот инструмент имеет некоторые недостатки. Например, большой массив может занимать неоправданно большой объем памяти. Кроме того, приходится увеличивать ряд на одно число за раз, поэтому нельзя выполнить цикл, например для последовательности четных чисел.

Что касается PHP 4.1, то значение переменной `$start` может быть больше значения переменной `$end`. В этом случае функция `range()` возвращает числа в убывающем порядке. Также можно использовать итерацию для последовательности символов:

```
print_r(range('l', 'p'));
Array
(
```

```
        [0] => l
        [1] => m
        [2] => n
        [3] => o
        [4] => p
    )
```

Цикл `for` использует только единственное целое и совершенно не работает с массивом. Возможности цикла `while` богаче, он предоставляет больший контроль над циклом, так как позволяет увеличивать и уменьшать переменную `$i` более свободно. Можно изменять переменную `$i` внутри цикла, что не всегда можно сделать с функцией `range()`, поскольку PHP читает весь массив при входе в цикл, и изменения в массиве не оказывают влияния на последовательность элементов.

См. также

Рецепт 4.3 для более подробной информации по инициализации массива рядом целых чисел; документацию по функции `range()` на *http://www.php.net/range*.

2.5. Генерация случайных чисел в пределах диапазона

Задача

Необходимо сгенерировать случайное число в пределах числового диапазона.

Решение

Для этого предназначена функция `mt_rand()`:

```
// случайное число между $upper и $lower, включительно
$random_number = mt_rand($lower, $upper);
```

Обсуждение

Генерация случайных чисел полезна, когда надо вывести на экран случайную картинку, случайным образом назначить стартовую точку в игре, выбрать случайную запись из базы данных или сгенерировать уникальный идентификатор сессии.

Для того чтобы сгенерировать случайное число в интервале между двумя точками, надо передать функции `mt_rand()` два аргумента:

```
$random_number = mt_rand(1, 100);
```

Вызов функции `mt_rand()` без аргументов возвращает число между нулем и максимальным случайным числом, возвращенным функцией `mt_getrandmax()`.

Компьютеру трудно сгенерировать действительно случайное число. Намного лучше он умеет методически следовать инструкциям и не так хорош, если от него требуются спонтанные действия. Если необходимо заставить компьютер выдать случайное число, то нужно дать ему определенный набор повторяемых команд, при этом сам факт повторяемости делает достижение случайности менее вероятным.

PHP имеет два различных генератора случайных чисел: классическую функцию под именем `rand()` и более совершенную функцию `mt_rand()`. MT (Mersenne Twister) – это генератор псевдослучайных чисел, названный в честь французского монаха и математика Марена Мерсенн (Marin Mersenne), исследовавшего простые числа. На этих простых числах и основан алгоритм данного генератора. Функция `mt_rand()` работает быстрее, чем функция `rand()`, и дает более случайные числа, поэтому мы отдаем предпочтение первой из них.

Если у вас версия PHP более ранняя, чем 4.2, то перед тем как первый раз вызвать функцию `mt_rand()` (или `rand()`), нужно инициализировать генератор начальным значением путем вызова функции `mt_srand()` (или `srand()`). Начальное значение – это число, которое случайная функция использует как основу для генерации возвращаемых ею случайных чисел; это относится к способу разрешения упомянутой выше дилеммы – повторяемость против случайности. В качестве начального значения, меняющегося очень быстро и с малой вероятностью повторяемости (именно этими свойствами должно характеризоваться хорошее начальное значение), можно взять значение, возвращенное высокоточной функцией времени `microtime()`. Генератор достаточно инициализировать один раз. PHP 4.2 и более поздних версий автоматически управляет инициализацией, но если начальное значение устанавливается вручную перед первым вызовом функции `mt_rand()`, то PHP не заменяет его своим собственным начальным значением.

Если нужно выбрать случайную запись из базы данных, то проще всего сначала определить общее количество полей в таблице, выбрать случайное число из этого диапазона, а затем запросить эту строку из базы данных:

```
$sth = $dbh->query('SELECT COUNT(*) AS count FROM quotes');
if ($row = $sth->fetchRow()) {
    $count = $row[0];
} else {
    die ($row->getMessage());
}

$random = mt_rand(0, $count - 1);

$sth = $dbh->query("SELECT quote FROM quotes LIMIT $random,1");
while ($row = $sth->fetchRow()) {
    print $row[0] . "\n";
}
```

Этот фрагмент кода определяет общее количество строк в таблице, генерирует случайное число из этого диапазона, а затем использует `LIMIT $random, 1` для выбора (`SELECT`) одной строки из таблицы, начиная с позиции `$random`.

В MySQL версии 3.23 или выше возможен альтернативный вариант:

```
$sth = $dbh->query('SELECT quote FROM quotes ORDER BY RAND() LIMIT 1');
while ($row = $sth->fetchRow()) {
    print $row[0] . "\n";
}
```

В этом случае MySQL сначала располагает строки в случайном порядке, а затем возвращает первую строку.

См. также

Рецепт 2.6 о том, как генерировать случайные числа со смещением; документацию по функции `mt_rand()` на *http://www.php.net/mt-rand* и по функции `rand()` на *http://www.php.net/rand*; MySQL-руководство по функции `RAND()` на *http://www.mysql.com/doc/M/a/Mathematical_functions.html*.

2.6. Генерация случайных чисел со смещением

Задача

Необходимо генерировать случайные числа, но с некоторым смещением, чтобы в определенном диапазоне числа появлялись чаще, чем в других. Например, нужно показать серию копий рекламных баннеров пропорционально количеству оставшихся копий каждой рекламной компании.

Решение

Используйте функцию `pc_rand_weighted()`, показанную в примере 2.1.

Пример 2.1. pc_rand_weighted( )

```
// возвращает взвешенный случайно выбранный ключ
function pc_rand_weighted($numbers) {
    $total = 0;
    foreach ($numbers as $number => $weight) {
        $total += $weight;
        $distribution[$number] = $total;
    }
    $rand = mt_rand(0, $total - 1);
    foreach ($distribution as $number => $weights) {
        if ($rand < $weights) { return $number; }
    }
}
```

Обсуждение

Представьте, что вместо массива, значения элементов которого отражают количество оставшихся копий объявлений, есть массив объявлений, в котором каждое объявление встречается ровно столько раз, сколько осталось его копий. Можно просто указать на не взвешенное случайное место внутри массива, и это будет реклама для показа. Вместо этого можно определить величину возможного массива (путем подсчета остающихся копий), выбрать случайное число из диапазона размера воображаемого массива, а затем пробежаться по массиву, определяя, какое объявление соответствует выбранному числу. Например:

```
$ads = array('ford' => 12234, // рекламодатель, остающиеся копии
             'att'  => 33424,
             'ibm'  => 16823);

$ad = pc_rand_weighted($ads);
```

См. также

Рецепт 2.5 о том, как генерировать случайные числа внутри диапазона.

2.7. Взятие логарифмов

Задача

Необходимо взять логарифм числа.

Решение

Для логарифмов по основанию *e* (натуральный логарифм) применяется функция `log()`:

```
$log = log(10);          // 2.30258092994
```

Логарифмы по основанию 10 вычисляются при помощи функции `log10()`:

```
$log10 = log10(10);      // 1
```

Для вычисления логарифмов по другим основаниям предназначена функция `pc_logn()`:

```
function pc_logn($number, $base) {
    return log($number) / log($base);
}

$log2  = pc_logn(10, 2); // 3.3219280948874
```

Обсуждение

И функция `log()`, и функция `log10()` определены только для положительных чисел. В функции `pc_logn()` базовая формула изменена и лога-

рифм числа по основанию `n` равен логарифму этого числа по произвольному основанию, поделенному на логарифм числа `n` по тому же самому основанию.

См. также

Документацию по функции `log()` на *http://www.php.net/log* и по функции `log10()` на *http://www.php.net/ log10*.

2.8. Вычисление степеней

Задача

Необходимо возвести число в степень.

Решение

Число *e* возводится в степень при помощи функции `exp()`:

```
$exp = exp(2);        // 7.3890560989307
```

Для возведения числа в произвольную степень предназначена функция `pow()`:

```
$exp = pow( 2, M_E);  // 6.5808859910179

$pow = pow( 2, 10);   // 1024
$pow = pow( 2, -2);   // 0.25
$pow = pow( 2, 2.5);  // 5.6568542494924

$pow = pow(-2, 10);   // 1024
$pow = pow( 2, -2);   // 0.25
$pow = pow(-2, -2.5); // NAN (Ошибка: Нечисло)
```

Обсуждение

Встроенная константа `M_E` — это приближение числа *e*. Она равна 2,7182818284590452354. Поэтому значения `exp($n)` и `pow(M_E, $n)` идентичны.

Функции `exp()` и `pow()` позволяют без труда создать очень большое число; если вы превысили максимальное значение числа в PHP (примерно 1.8e308), то обратитесь к рецепту 2.13, описывающему применение функций с произвольно выбираемой точностью. Эти функции PHP возвращают `INF`, бесконечность, если результат слишком большой, и `NAN`, нечисло, в случае ошибки.

См. также

Документацию по функции `pow()` на *http://www.php.net/pow*, по функции `exp()` на *http://www.php.net/exp* и информацию по предопределенным математическим константам на *http://www.php.net/math*.

2.9. Форматирование чисел

Задача

Необходимо напечатать число с разделителями тысяч и десятков тысяч. Например, нужно вывести стоимость покупок в магазинной тележке.

Решение

Функция `number_format()` позволяет вывести число в формате целого:

```
$number = 1234.56;
print number_format($number);    // 1,235 поскольку число округлено
```

Определите число десятичных разрядов для форматирования в виде десятичной дроби:

```
print number_format($number, 2); // 1,234.56
```

Обсуждение

Функция `number_format()` форматирует число, вставляя необходимые разделители десятков и тысяч в соответствии с локализацией. Если требуется установить эти значения вручную, передайте их как третий и четвертый параметры:

```
$number = 1234.56;
print number_format($number, 2, '@', '#'); // 1#234@56
```

Третий аргумент выступает в качестве десятичной точки, а последний отделяет тысячи. Эти два аргумента необходимо указывать вместе.

По умолчанию функция `number_format()` округляет число до ближайшего целого. Если надо сохранить все число, но заранее неизвестно, сколько разрядов будет после десятичной точки, прибегают к следующему приему:

```
$number = 1234.56; // ваше число
list($int, $dec) = explode('.', $number);
print number_format($number, strlen($dec));
```

См. также

Документацию по функции `number_format()` на *http://www.php.net/number-format*.

2.10. Правильная печать слов во множественном числе

Задача

Необходимо правильно выбрать число – единственное или множественное – в зависимости от значения переменной. Например, вы воз-

вращаете текст, который зависит от количества совпадений, найденных при поиске.

Решение

Это делается при помощи условного выражения:

```
$number = 4;
print "Your search returned $number " . ($number == 1 ? 'hit' : 'hits') . '.';
Your search returned 4 hits.
```

Обсуждение

Можно записать эту строку немного короче:

```
print "Your search returned $number hit" . ($number == 1 ? '' : 's') . '.';
```

Однако в других случаях образования множественного числа, таких как «person» → «people», очевидно, что надо изменить все слово, а не одну букву.

Есть другой вариант – вызывать одну функцию для всех случаев образования множественного числа, как показано в функции `pc_may_pluralize()` из примера 2.2.

Пример 2.2. pc_may_pluralize( )

```
function pc_may_pluralize($singular_word, $amount_of) {

    // массив особых слов во множественном числе
    $plurals = array(
        'fish' => 'fish',
        'person' => 'people',
    );

    // единственное значение
    if (1 == $amount_of) {
        return $singular_word;
    }

    // более одного, особая форма множественного числа
    if (isset($plurals[$singular_word])) {
        return $plurals[$singular_word];
    }

    // более одного, обычная форма множественного числа:
    // добавить 's' в конце слова
    return $singular_word . 's';
}
```

Примеры:

```
$number_of_fish = 1;
print "I ate $number_of_fish " . pc_may_pluralize('fish',
                                                  $number_of_fish) . '.';
```

```
$number_of_people = 4;
print 'Soylent Green is ' . pc_may_pluralize('person',
                                          $number_of_people) . '!';
I ate 1 fish.
Soylent Green is people!
```

Если в коде предполагается наличие нескольких слов во множественном числе, то нужна функция, облегчающая чтение, такая как `pc_may_pluralize()`. Этой функции передается слово в единственном числе в качестве первого аргумента и количество включений в качестве второго аргумента. В функцию включен большой массив, `$plurals`, содержащий все особые случаи. Если переменная `$amount` равна `1`, то функция возвращает оригинальное слово. Если переменная больше единицы, то возвращается слово в особой форме множественного числа, если такая существует. По умолчанию добавляется только «s» в конце слова.[1]

2.11. Вычисление тригонометрических функций

Задача

Необходимо применить тригонометрические функции, такие как синус, косинус и тангенс.

Решение

В PHP реализованы тригонометрические функции `sin()`, `cos()` и `tan()`:

```
$cos = cos(2.1232);
```

А также обратные им функции `asin()`, `acos()` и `atan()`:

```
$atan = atan(1.2);
```

Обсуждение

Эти функции принимают аргументы в радианах, а не в градусах. (В случае затруднений см. рецепт 2.12.)

Функция `atan2()` принимает две переменные `$x` and `$y` и вычисляет `atan($x/$y)`. Однако она всегда возвращает правильный знак, поскольку определяет квадрант результата по значениям обоих параметров.

Для секанса, косеканса и котангенса необходимо вручную вычислить обратные значения функций `sin()`, `cos()` и `tan()`:

```
$n = .707;

$secant    = 1 / sin($n);
```

[1] Естественно, сказанное относится к словам английского языка. – *Примеч. ред.*

```
$cosecant  = 1 / cos($n);
$cotangent = 1 / tan($n);
```

Начиная с PHP 4.1 доступны гиперболические функции: `sinh()`, `cosh()` и `tanh()`, а также `asin()`, `cosh()` и `atanh()`. Однако обратные функции не поддерживаются в Windows.

См. также

Рецепт 2.12 о выполнении тригонометрических операций в градусах, а не в радианах; документацию по функции `sin()` на *http://www.php.net/sin*, по функции `cos()` на *http://www.php.net/cos*, по функции `tan()` на *http://www.php.net/tan*, по функции `asin()` на *http://www.php.net/asin*, по функции `acos()` на *http://www.php.net/acos*, по функции `atan()` на *http://www.php.net/atan* и по функции `atan2()` на *http://www.php.net/atan2*.

2.12. Тригонометрические вычисления не в радианах, а в градусах

Задача

Необходимо применить тригонометрические функции к значениям, выраженным в градусах.

Решение

Примените функцию `deg2rad()` и функцию `rad2deg()` к вводу и выводу:

```
$cosine = rad2deg(cos(deg2rad($degree)));
```

Обсуждение

По определению 360 градусов равны 2π радиан, поэтому данное преобразование нетрудно произвести вручную. Эти функции берут встроенное в PHP значение числа π, поэтому можно гарантировать высокую точность результата. Для других операций с этим числом вполне подходит константа `M_PI`, равная 3.14159265358979323846.

Для градиан[1] (gradians) встроенная функция пока не написана. Это не ошибка, а политика разработчиков.

См. также

Рецепт 2.12 по основам тригонометрии; документацию по функции `deg2rad()` на *http://www.php.net/deg2rad* и по функции `rad2deg()` на *http://www.php.net/rad2deg*.

---

[1] 360 градусов равны 400 градиан. – *Примеч. ред.*

2.13. Работа с очень большими и очень маленькими числами

Задача

Необходимо работать с числами, выходящими из дапазона допустимых в PHP значений чисел с плавающей точкой.

Решение

Для этого нужна либо библиотека BCMath, либо библиотека GMP.

Применение BCMath:

```
$sum = bcadd('1234567812345678', '8765432187654321');

// переменная $sum равна теперь '9999999999999999'
print $sum;
```

Применение GMP:

```
$sum = gmp_add('1234567812345678', '8765432187654321');

// $sum теперь ресурс GMP, а не строка; для преобразования
// используйте функцию gmp_strval()
print gmp_strval($sum);
```

Обсуждение

Библиотека BCMath проста в применении. Числа передаются как строки, а функция возвращает сумму (или разность, произведение и т. д.) в виде строки. Однако набор действий, которые можно производить над числами с помощью библиотеки BCMath, ограничен основными арифметическими операциями.

Библиотека GMP доступна начиная с версии PHP 4.0.4. Большинство представителей семейства функций библиотеки GMP в качестве аргументов принимают целые и строки, но они преимущественно обмениваются числами в виде ресурсов, которые, по сути дела, представляют собой ссылки на числа. Поэтому, в противоположность функциям BCMath, которые возвращают строки, функции GMP возвращают только ресурсы. Последние передаются затем любой функции GMP, которая работает с ними как с числами.

Единственной оборотной стороной медали является то, что при работе с не-GMP функциями необходимо непосредственно конвертировать ресурсы с помощью функции `gmp_strval()` или функции `gmp_intval()`.

Функции GMP либерально относятся к входным параметрам. Например:

```
$four = gmp_add(2, 2);             // Передаем целые
$eight = gmp_add('4', '4');        // Или строки
$twelve = gmp_add($four, $eight);  // Или ресурсы GMP
print gmp_strval($twelve);         // Печатаем 12
```

Впрочем, с числами GMP можно совершать множество операций помимо сложения, таких как возведение в степень, быстрое вычисление больших факториалов, нахождение наибольшего общего делителя (НОД) и других:

```
// Возведение числа в степень
$pow = gmp_pow(2, 10);              // 1024

// Быстрое вычисление больших факториалов
$factorial = gmp_fact(20);          // 2432902008176640000

// Нахождение НОД
$gcd = gmp_gcd (123, 456);          // 3

// Другой нестандартный математический инструментарий
$legdendre = gmp_legendre(1, 7);    // 1
```

Библиотеки BCMath и GMP не обязательно доступны во всех конфигурациях PHP. Начиная с версии PHP 4.0.4 библиотека BCMath связана с PHP, поэтому она, вероятно, должна быть легко доступна. Однако если библиотека GMP не связана с PHP, то необходимо ее загрузить, инсталлировать и в процессе конфигурирования проинструктировать PHP об использовании этой библиотеки. Проверьте значения функций `function_defined('bcadd')` и `function_defined('gmp_init')` чтобы определить, можно ли использовать библиотеки BCMath и GMP.

См. также

Документацию по библиотеке BCMath на *http://www.php.net/bc* и по библиотеке GMP на *http://www.php.net/ gmp*.

2.14. Преобразование из одной системы счисления в другую

Задача

Необходимо преобразовать число из одной системы счисления в другую.

Решение

Обратитесь к функции `base_convert()`:

```
$hex = 'a1';                    // шестнадцатеричное число (основание 16)

// преобразование из основания 16 в основание 10
$decimal = base_convert($hex, 16, 10); // переменная $decimal теперь равна 161
```

Обсуждение

Функция `base_convert()` изменяет строку в одной системе в соответствующую строку в другой системе. Она работает для всех систем с основаниями от 2 до 36 включительно. Для изображения чисел в системах

с основанием больше 10 в качестве дополнительных символов используются буквы от a до z. Первый аргумент – это число, которое нужно преобразовать, за ним следует основание его системы, а в конце – основание ситемы, в которую требуется преобразовать число.

Существует несколько специальных функций для прямого и обратного преобразования чисел в десятичную систему из других наиболее востребованных систем с основаниями 2, 8 и 16. Это функции bindec() и decbin(), octdec() и decoct(),hexdec() и dechex():

```
// преобразование в десятичную систему
print bindec(11011); // 27
print octdec(33);    // 27
print hexdec('1b');  // 27

// преобразование из десятичной системы
print decbin(27);    // 11011
print decoct(27);    // 33
print dechex(27);    // 1b
```

Есть и другой вариант – можно обратиться к функции sprintf(), позволяющей преобразовывать десятичные числа в двоичные, восьмеричные и шестнадцатеричные и предоставляющей широкие возможности форматирования, например с нулями в начале числа и возможностью выбора между верхним и нижним регистром при отображении шестнадцатеричных чисел.

Пусть требуется вывести на печать значения цветов HTML:

```
printf('#%02X%02X%02X', 0, 102, 204); // #0066CC
```

См. также

Документацию по функции base_convert() на *http://www.php.net/base-convert* и по опциям форматирования функции sprintf() на *http://www.php.net/sprintf*.

2.15. Вычисления с недесятичными числами

Задача

Необходимо выполнить математические операции не над десятичными числами, а над восьмеричными или шестнадцатеричными. Например, определить корректные цвета веб-сайта в шестнадцатеричном формате.

Решение

Предваряйте число начальным символом, чтобы PHP смог узнать, что это не десятичное число. Следующие значения равны:

```
0144  // основание 8
 100  // основание 10
0x64  // основание 16
```

Ниже показан отсчет от 1 до 15 в шестнадцатеричной нотации:

```
for ($i = 0x1; $i < 0x10; $i++) { print "$i\n"; }
```

Обсуждение

Даже если в цикле `for` используются числа в шестнадцатеричном формате, по умолчанию все числа печатаются в десятичном формате. Другими словами, код из предыдущего раздела «Решение» не печатает «..., 8, 9, a, b, ...». Напечатать число в шестнадцатеричном формате можно при помощи одного из методов, перечисленных в рецепте 2.14. Например:

```
for ($i = 0x1; $i < 0x10; $i++) { print dechex($i) . "\n"; }
```

Большинство вычислений проще выполнять в десятичной системе. Однако иногда логичнее переключиться на систему с другим основанием, например при использовании 216 веб-корректных цветов. Каждый код веб-цвета представляется в виде `RRGGBB`, где `RR` – это красный цвет, `GG` – зеленый цвет, а `BB` – голубой. Каждый цвет на самом деле представляет собой двузначное шестнадцатеричное число от 0 до FF.

Особыми веб-цвета делает то, что каждый из кодов `RR`, `GG` и `BB` должен быть одним из шести чисел: 00, 33, 66, 99, CC и FF (в десятичном формате: 0, 51, 102, 153, 204, 255). Поэтому 003366 – это веб-корректный цвет, а 112233 – нет. Веб-корректные цвета отображаются на 256-цветном мониторе без сглаживания переходов.

В приведенном ниже тройном цикле числа создаваемого списка записываются в шестнадцатеричной системе, чтобы подчеркнуть шестнадцатеричную природу списка:

```
for ($rr = 0; $rr <= 0xFF; $rr += 0x33)
    for ($gg = 0; $gg <= 0xFF; $gg += 0x33)
        for ($bb = 0; $bb <= 0xFF; $bb += 0x33)
            printf("%02X%02X%02X\n", $rr, $gg, $bb);
```

В данном цикле вычисляются все веб-корректные цвета. Пошаговое приращение записывается в шестнадцатеричной системе, поскольку это усиливает шестнадцатеричную связь между числами. Печатайте их с помощью функции `printf()`, чтобы отформатировать их в виде шестнадцатеричных чисел в верхнем регистре длиной как минимум в две цифры. Число с одной цифрой отображается с нулем в начале.

См. также

Более подробную информацию о преобразовании чисел в различные системы счисления в рецепте 2.14; главу 3 «Web Design Principles for Print Designers» (Принципы Web-дизайна для дизайнеров печатных изданий) в книге «Web Design in a Nutshell» (O'Reilly).[1]

[1] Нидерст Дж. «Web-мастеринг для профессионалов». – Пер. с англ. – СПб: Питер, 2000.

3

Дата и время

3.0. Введение

На первый взгляд, отображение и обработка дат и времени кажется простой задачей, но иногда она усложняется в зависимости от многочисленности ваших пользователей и сложности их требований. Рассеяны ли пользователи по более чем одной временной зоне? Вероятно, да, только если вы не строите интранет или сайт с очень специфической географией аудитории. Смутит ли вашу аудиторию метка даты/времени вида «2002-07-20 14:56:34 EDT» или она предпочтет знакомое представление, например «20 июля 2002 года, 14:56». Определить количество часов в промежутке между 10 часами сегодняшнего дня и 19 часами завтрашнего довольно легко. А между 3-мя часами сегодняшнего утра и полуднем первого дня следующего месяца? Определение разности между датами рассмотрено в рецептах 3.5 и 3.6.

Эти вычисления и обработка становятся еще более раздражающими из-за перехода на летнее время (Daylight Saving Time, DST). При этом появляется время, которое никогда не наступает (в большинстве американских штатов это время с 2 до 3 часов ночи в первое воскресенье апреля), и время, которое наступает дважды (в большинстве американских штатов это время с 1 до 2 часов ночи в последнее воскресенье октября). Некоторые из ваших пользователей живут в регионах, в которых соблюдается DST, а некоторые нет. Способы работы с часовыми поясами и DST рассмотрены в рецептах 3.11 и 3.12.

Два соглашения делают программную обработку времени значительно более простой. Во-первых, в качестве внутреннего представления времени следует использовать согласованное универсальное время (UTC, Coordinated Universal Time), известное еще как GMT, Greenwich Mean Time – среднее время по Гринвичу) – патриарх семейства часовых поясов, не учитывающий DST (переход на летнее время). Это часовой пояс нулевого градуса долготы, а все остальные часовые пояса определяются как смещение (положительное или отрицательное) относительно

него. Во-вторых, время надо рассматривать не как массив различных значений для месяца, дня, минуты, секунды и т. д., а как количество секунд, истекшее с начала эпохи UNIX: полночи 1 января 1970 года (конечно, UTC). Это значительно упрощает вычисление интервалов времени, и PHP предоставляет множество функций, помогающих легко переходить от меток времени UNIX к понятным человеку представлениям времени и обратно.

Функция `mktime()` вырабатывает метку времени UNIX из данного набора компонентов времени, тогда как функция `date()` принимает метку времени, а возвращает форматированную строку даты и времени. Посредством этих функций можно, например, определить, на какой день недели пришелся новогодний праздник в 1986 году:

```
$stamp = mktime(0,0,0,1,1,1986);
print date('l',$stamp);
Wednesday
```

В данном случае функция `mktime()` возвращает метку времени UNIX в полночь 1 января 1986 года. Символ форматирования `l` в функции `date()` означает, что она вернет полное название дня недели, соответствующее данной метке времени. Многие символы форматирования, доступные в функции `date()`, подробно описаны в рецепте 3.4.

В этой книге фраза *метка времени UNIX* относится к количеству секунд, истекших с начала эпохи UNIX. *Части времени* (или *части даты* или *части времени и даты*) означают массив или группу компонентов времени и даты, таких как день, месяц, год, час, минута и секунда. *Форматированная строка времени* (или *форматированная строка даты* и т. д.) означает строку, содержащую некоторым образом сгруппированные части времени и даты, например «2002-03-12», «Wednesday, 11:23 A.M.», или «February 25».

Если вы использовали метку времени UNIX в качестве внутреннего представления времени, то смогли избежать любых последствий, связанных с проблемой 2000 года, поскольку разность между 946702799 (1999-12-31 23:59:59 UTC) и 946702800 (2000-01-01 00:00:00 UTC) трактуется точно так же, как и разность между любыми двумя метками даты/времени. Однако можно столкнуться с проблемой 2038 года. 19 января 2038 года в 3:14:07 A.M. (UTC) – это 2147483647 секунд после полуночи 1 января 1970 года. Что особенного в 2147483647? Это $2^{31} - 1$, максимально возможное целое число со знаком в 32-битном представлении (32-й бит представляет знак.)

Решение? Выберите время до 19 января 2038 года для покупки оборудования, которое, скажем, использует 64 бита для хранения значений времени. Таким образом, вы купите еще примерно 292 триллиона лет. (Всего 39 бит позволили бы вам протянуть примерно до 10680 года и удачно пережить потрясения, вызванные ошибкой десятитысячного года, которая сравняла с землей фабрики погоды и станции сверхсве-

товых путешествий.) 2038 год может показаться сейчас слишком далеким, но таким же далеким казался и 2000-й год программистам на КОБОЛе в 1950-х и 1960-х годах. Не повторяйте этой ошибки!

3.1. Определение текущей даты и времени

Задача

Необходимо узнать текущее время или дату.

Решение

Для получения отформатированной строки времени предназначена функция `strftime()` или `date()`:

```
print strftime('%c');
print date('r');
Mon Aug 12 18:23:45 2002
Mon, 12 Aug 2002 18:23:45 -0400
```

Функции `getdate()` и `localtime()` позволяют получить отдельные части времени:

```
$now_1 = getdate();
$now_2 = localtime();
print "$now_1[hours]:$now_1[minutes]:$now_1[seconds]";
print "$now_2[2]:$now_2[1]:$now_2[0]";
18:23:45
18:23:45
```

Обсуждение

Функции `strftime()` и `date()` могут выработать множество отформатированных строк времени или даты и рассматриваются в рецепте 3.4. Напротив, и функция `localtime()`, и функция `getdate()` возвращают массив, элементами которого являются отдельные части даты и времени.

Ассоциативный массив, который возвращает функция `getdate()`, содержит пары ключ/значение, перечисленные в табл. 3.1.

Таблица 3.1. Массив, возвращаемый функцией getdate( )

Key	Value
seconds	Секунды
minutes	Минуты
hours	Часы
mday	День месяца
wday	День недели, числовое значение (воскресенье – это 0, суббота – это 6)

Таблица 3.1 (продолжение)

Key	Value
mon	Месяц, числовое значение
year	Год, числовое значение
yday	День года, числовое значение (т. е. 299)
weekday	День недели, полное текстовое значение (т. е. «Friday»)
month	Месяц, полное текстовое значение (т. е. «January»)

Следующий пример показывает, как использовать функцию getdate() для вывода на печать месяца, дня и года.[1]

```
$a = getdate();
printf('%s %d, %d',$a['month'],$a['mday'],$a['year']);
August 7, 2002
```

Передайте фунции getdate() метку времени UNIX в качестве аргумента, чтобы обеспечить соответствие значений массива локальному времени данной временной метки. Например, месяц, день и год, соответствующие метке времени UNIX, равной 163727100, это:

```
$a = getdate(163727100);
printf('%s %d, %d',$a['month'],$a['mday'],$a['year']);
March 10, 1975
```

Функция localtime() возвращает массив частей времени и даты. Кроме того, она принимает метку времени UNIX в качестве необязательного первого аргумента, а также логическое значение в качестве необязательного второго аргумента. Если этот второй аргумент равен true, то функция localtime() возвращает ассоциативный массив вместо массива с числовым индексом. Ключи этого массива совпадают с членами

---

[1] Для того чтобы название месяца или дня недели выводилось на русском языке, следует установить нужную локализацию с помощью функции setlocale(). Например, для кодировки Windows-1251 можно попробовать написать:

```
setlocale(LC_ALL,"ru_RU.CP1251");
echo strftime(«F»);
```

Однако такая локализация должна присутствовать в системе (подробнее см. *http://www.php.net/setlocale* и документацию по операционной системе). Если не удается получить результат таким способом, то можно выводить названия месяцев и дней недели вручную:

```
$rus_months=array(1=>'января', 'февраля', 'марта', 'апреля', 'мая',
'июня', 'июля', августа', 'сентября','октября','ноября','декабря');
echo date('d') ." ". $rus_months[date('n')] ." ". date('Y');
```

– *Примеч. науч. ред.*

структуры tm_struct, возвращаемой функцией языка C localtime(), как показано в табл. 3.2.

Таблица 3.2. Массив, возвращаемый фунцией localtime( )

Числовая позиция	Ключ	Значение
0	tm_sec	Секунды
1	tm_min	Минуты
2	tm_hour	Часы
3	tm_mday	День месяца
4	tm_mon	Месяц года (Январь, если 0)
5	tm_year	Год с 1900 года
6	tm_wday	День недели
7	tm_yday	День года
8	tm_isdst	Учитывается ли переход на летнее время (DST)?

Следующий пример показывает, как использовать функцию localtime() для вывода на печать текущей даты в формате месяц/число/год:

```
$a = localtime();
$a[4] += 1;
$a[5] += 1900;
print "$a[4]/$a[3]/$a[5]";
8/7/2002
```

Перед выводом на печать значение месяца увеличивается на 1, так как функция localtime() начинает отсчет месяцев с 0 для января, а мы хотим видеть 1, если текущий месяц январь. Таким же образом год увеличивается на 1900, поскольку функция localtime() начинает отсчет лет с 0 для 1900-го.

Подобно getdate(), функция localtime() принимает метку времени UNIX в качестве необязательного первого аргумента и возвращает части времени для этой временной метки:

```
$a = localtime(163727100);
$a[4] += 1;
$a[5] += 1900;
print "$a[4]/$a[3]/$a[5]";
3/10/1975
```

См. также

Документацию по функции strftime() на *http://www.php.net/strftime*, по функции date() на *http://www.php.net/date*, по функции getdate() на *http://www.php.net/getdate* и по функции localtime() на *http://www.php.net/localtime*.

3.2. Преобразование времени и частей времени в метку времени UNIX

Задача

Необходимо определить, какая метка времени UNIX соответствует множеству частей времени и даты.

Решение

Если части времени и даты относятся к локальной временной зоне, то следует применять функцию `mktime()`:

```
// 7:45:03 PM on March 10, 1975, local time
$then = mktime(19,45,3,3,10,1975);
```

Функция `gmmktime()`, если части времени и даты относятся к часовому поясу GMT:

```
// 7:45:03 PM on March 10, 1975, in GMT
$then = gmmktime(19,45,3,3,10,1975);
```

Для получения текущих времени и даты в локальной зоне или в зоне UTC никаких аргументов передавать не надо:

```
$now = mktime();
$now_utc = gmmktime();
```

Обсуждение

Функции `mktime()` и `gmmktime()` принимают части даты и времени (час, минуту, секунду, месяц, день, год, флаг DST) и возвращают соответствующую метку даты/времени эпохи UNIX. Компоненты рассматриваются функцией `mktime()`, как локальное время, а функция `gmmktime()` считает их датой и временем в зоне UTC. Для обеих функций седьмой аргумент, флаг DST (1, если DST учитывается, и 0, если нет) необязателен. Эти функции возвращают осмысленные результаты только для времени, принадлежащего эпохе UNIX. Большинство систем хранят метку времени в 32-битном целом со знаком, поэтому «принадлежащее эпохе» означает время между 8:45:51 P.M. 13 декабря 1901 года UTC и 3:14:07 A.M. 19 января 2038 года UTC.

В следующем примере переменная `$stamp_now` содержит метку времени в момент вызова функции `mktime()`, а переменная `$stamp_future` – метку времени для 3:25 P.M. 4 июня 2012 года:

```
$stamp_now = mktime();
$stamp_future = mktime(15,25,0,6,4,2012);

print $stamp_now;
print $stamp_future;
1028782421
1338837900
```

Обе метки времени могут быть переданы обратно в функцию `strftime()` для получения форматированной строки времени:

```
print strftime('%c',$stamp_now);
print strftime('%c',$stamp_future);
Thu Aug  8 00:53:41 2002
Mon Jun 4 15:25:00 2012
```

Приведенные выше вызовы функции `mktime()` были сделаны на компьютере, находящемся в зоне EDT (которая на четыре часа отстает от зоны GMT), поэтому, если вместо нее вызвать функцию `gmmktime()`, будет получена метка времени, на 14400 секунд (четыре часа) меньшая:

```
$stamp_now = gmmktime();
$stamp_future = gmmktime(15,25,0,6,4,2012);

print $stamp_now;
print $stamp_future;
1028768021
1338823500
```

Передавая эту, сгенерированную функцией `gmmktime()`, метку времени обратно в функцию `strftime()`, получаем форматированную строку времени, которая также отстает на четыре часа:

```
print strftime('%c',$stamp_now);
print strftime('%c',$stamp_future);
Wed Aug  7 20:53:41 2002
Mon Jun 4 11:25:00 2012
```

См. также

Рецепт 3.3, описывающий преобразование метки времени обратно в части времени и даты; документацию по функции `mktime()` на *http://www.php.net/mktime* и по функции `gmmktime()` на *http://www.php.net/gmmktime*.

3.3. Преобразование метки времени в части времени и даты

Задача

Необходимо определить части времени и даты, соответствующие метке времени.

Решение

Передайте метку времени функции `getdate()`:

```
$time_parts = getdate(163727100);
```

Обсуждение

Части времени, возвращенные функцией `getdate()`, подробно рассматриваются далее. Эти части времени представляют местное время.

Получение компонентов времени в другом часовом поясе, соответствующем определенной метке времени, описано в рецепте 3.11.

См. также

Рецепт 3.2 о преобразовании частей времени и даты обратно в метку времени; рецепт 3.11 о работе с часовыми поясами; документацию по функции `getdate()` на *http://www.php.net/ getdate*.

3.4. Вывод на печать даты и времени в определенном формате

Задача

Необходимо вывести на печать дату и время, отформатированные определенным образом.

Решение

Это делается при помощи функции `date()` или функции `strftime()`:

```
print strftime('%c');
print date('m/d/Y');
Tue Jul 30 11:31:08 2002
07/30/2002
```

Обсуждение

Функции `date()` и `strftime()` отличаются гибкостью и способны вырабатывать форматированную строку времени с множеством компонент. В табл. 3.3 перечислены символы форматирования, поддерживаемые этими функциями. В столбце Windows показано, поддерживает ли функция `strftime()` символ форматирования в системе Windows.

Таблица 3.3. Символы форматирования функций strftime( ) and date( )

Тип	strftime()	date ()	Описание	Диапазон	Windows
Час	%H	H	Час, числовое значение, 24-часовой формат часов	00–23	Да
Час	%I	h	Час, числовое значение, 12-часовой формат часов	01–12	Да
Час	%k		Час, числовое значение, 24-часовой формат часов, начальный нуль в качестве пробела	0–23	Нет

Тип	strftime()	date ()	Описание	Диапазон	Windows
Час	%l		Час, числовое значение, 12-часовой формат часов, начальный нуль в качестве пробела	1–12	Нет
Час	%p	A	AM или PM обозначение для текущей локализации		Да
Час	%P	a	am/pm обозначение для текущей локализации		Нет
Час		G	Час, числовое значение, 24-часовой формат часов, начальный нуль исключен	0–23	Нет
Час		g	Час, числовое значение, 12-часовой формат часов, начальный нуль исключен	0–1	Нет
Минута	%M	I	Минута, числовое значение	00–59	Да
Секунда	%S	s	Секунда, числовое значение	00–61[a]	Да
День	%d	d	День месяца, числовое значение	01–31	Да
День	%e		День месяца, числовое значение, начальный нуль в качестве пробела	1–31	Нет
День	%j	z	День года, числовое значение	001–366 для strftime(); 0–365 для date( )	Да
День	%u		День недели, числовое значение (Понедельник – это 1)	1–7	Нет
День	%w	w	День недели, числовое значение (Воскресенье – это 0)	0–6	Да
День		j	День месяца, числовое значение, начальный нуль исключен	1–31	Нет
День		S	Английский порядковый суффикс для дня месяца, текстовое значение	«st», «th», «nd», «rd»	Нет

[a] Диапазон секунд расширен до 61, чтобы учесть потерянные секунды.

Таблица 3.3 (продолжение)

Тип	strftime()	date ()	Описание	Диапазон	Windows
Неделя	%a	D	Сокращенное имя дня недели, текст для текущей локализации		Да
Неделя	%A	l	Полное имя дня недели, текст для текущей локализации		Да
Неделя	%U		Номер недели в году; числовое значение; первое воскресенье – это первый день первой недели	00–53	Да
Неделя	%V	W	ISO 8601:1988 номер недели в году; числовое значение; неделя 1 – это первая неделя, в которой минимум 4 дня в текущем году; понедельник – это первый день недели	01–53	Нет
Неделя	%W		Номер недели в году; числовое значение; первый понедельник – это первый день первой недели	00–53	Да
Месяц	%B	F	Полное имя месяца, текст для текущей локализации		Да
Месяц	%b	M	Сокращенное имя месяца, текст для текущей локализации		Да
Месяц	%h		То же, что и %b		Нет
Месяц	%m	m	Месяц, числовое значение	01–12	Да
Месяц		n	Месяц, числовое значение, начальный нуль исключен	1–12	Нет
Месяц		t	Длина месяца в днях, числовое значение	28, 29, 30, 31	Нет
Год	%C		Век, числовое значение	00–99	Нет
Год	%g		Такое же как %G, но без века	00–99	Нет
Год	%G		ISO 8601 год с веком; числовое значение; год из четырех цифр, соответствующий неделе в формате ISO число; то же, что и %y, за исключением того, что если неделя в формате ISO относится к предыдущему или следующему году, то берется именно этот год		Нет

Тип	strftime()	date ()	Описание	Диапазон	Windows
Год	%y	y	Год без века, числовое значение	00–99	Да
Год	%Y	Y	Год, числовое значение, включая век		Да
Год		L	Флаг високосного года (True соответствует 1)	0, 1	Нет
Часовой пояс	%z	O	Смещение от GMT в часах, +/–HHMM (т. е. –0400, +0230)	–от 1200 до +1200	Да, но действует подобно %Z
Часовой пояс	%Z	T	Часовой пояс, имя, или сокращение; текстовое значение		Да
Часовой пояс		I	Флаг летнего времени (True соответствует 1)	0, 1	Нет
Часовой пояс		Z	Смещение от GMT в секундах; на запад от GMT – отрицательное, на восток от GMT – положительное	от –43200 до 43200	Нет
Составной	%c		Стандартный формат даты и времени для местной локализации		Да
Составной	%D		То же, что и %m/%d/%y		Нет
Составной	%F		То же, что и %Y-%m-%d		Нет
Составной	%r		Время в нотации AM или PM для местной локализации		Нет
Составной	%R		Время в 24-часовой нотации для местной локализации		Нет
Составной	%T		Время в 24-часовой нотации (то же, что и %H:%M:%S)		Нет
Составной	%x		Стандартный формат даты для местной локализации (без времени)		Да
Compound	%X		Стандартный формат времени для местной локализации (без даты)		Да
Составной		r	Дата в формате RFC 822 (т. е. «Thu, 22 Aug 2002 16:01:07 +0200»)		Нет
Other	%s	U	Секунды с начала эпохи		Нет

Таблица 3.3 (продолжение)

Тип	strftime()	date ()	Описание	Диапазон	Windows
Other		B	Новое универсальное время (Swatch Internet time)		Нет
Форматирование	%%		Символьная константа %		Да
Форматирование	%n		Символ новой строки		Нет
Форматирование	%t		Символ табуляции		Нет

Первый аргумент каждой функции – это строка формата, а второй – метка времени. Если второй аргумент пропущен, то обе функции по умолчанию выводят текущие дату и время. Функции date() и strftime() работают с локальным временем, но каждая из них имеет своего UTC-двойника (gmdate() и gmstrftime() соответственно).

Символы форматирования функции date() уникальны для PHP, но функция strftime() использует функцию strftime() из библиотеки C и поэтому может оказаться более понятной тем, кто перешел на программирование в PHP с других языков, но в то же время, из-за этого ее поведение на разных платформах может несколько отличаться. Windows не поддерживает такое многообразие параметров функции strftime(), которое имеет большинство систем UNIX. Также функция strftime() предполагает, что каждый из ее символов форматирования начинается с символа % (вспомните функцию printf()), что упрощает выработку строки, включающей множество значений времени и даты.

Например, в 12:49 P.M. 15 июля 2002 года, код выдаст на печать:

```
It's after 12 pm on July 15, 2002
```

При этом применение функции strftime() может выглядеть следующим образом:

```
print strftime("It's after %I %P on %B %d, %Y");
```

А в случае применения функции date() это может выглядеть так:

```
print "It's after ".date('h a').' on '.date('F d, Y');
```

Не связанные с датой символы в строке формата хорошо подходят для функции strftime(), поскольку она ищет символ %, чтобы определить, куда вставить соответствующую информацию о времени. Однако функция date() не поддерживает такого ограничителя, поэтому единственное дополнение, которое можно вставить в строку форматирова-

ния, это символы пробела и пунктуации. Если вы передадите функции date() строку форматирования функции strftime():

```
print date("It's after %I %P on %B%d, %Y");
```

то, скорее всего, вам не понравится то, что вы получите:

```
131'44 pmf31eMon, 15 Jul 2002 12:49:44 -0400 %1 %P o7 %742%15, %2002
```

Для получения с помощью функции date() частей времени, которые легко вставлять в строку, сгруппируйте все части времени и даты, полученные от функции date(), в одну строку, вставляя между отдельными компонентами ограничитель, который функция date() никуда не передает, и который сам не является частью ни одной из подстрок. Затем с помощью функции explode() с этим символом-ограничителем, поместите каждую часть возвращенного функцией date() значения в массив, который легко вставить в строку вывода:

```
$ar = explode(':',date("h a:F d, Y"));
print "It's after $ar[0] on $ar[1]";
```

См. также

Документацию по функции date() на *http://www.php.net/date* и по функции strftime() на *http://www.php.net/strftime*; для систем UNIX, *man strftime* о параметрах функции strftime(), специфичных для вашей системы; для Windows – см. подробное описание функции strftime() на *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_strftime.2c_.wcsftime.asp*.

3.5. Определение разности между двумя датами

Задача

Необходимо определить время, прошедшее между двумя датами. Например, надо сообщить пользователю, сколько прошло времени с момента его последней регистрации на вашем сайте.

Решение

Преобразуйте обе даты в метки времени и вычтите одну из другой. Приведенный ниже код позволяет выделить из результата недели, дни, часы, минуты и секунды:

```
// 7:32:56 pm 10 мая 1965 года
$epoch_1 = mktime(19,32,56,5,10,1965);
// 4:29:11 am 20 ноября 1962 года
$epoch_2 = mktime(4,29,11,11,20,1962);

$diff_seconds  = $epoch_1 - $epoch_2;
$diff_weeks    = floor($diff_seconds/604800);
$diff_seconds -= $diff_weeks   * 604800;
```

```
$diff_days     = floor($diff_seconds/86400);
$diff_seconds -= $diff_days    * 86400;
$diff_hours    = floor($diff_seconds/3600);
$diff_seconds -= $diff_hours   * 3600;
$diff_minutes  = floor($diff_seconds/60);
$diff_seconds -= $diff_minutes * 60;

print "The two dates have $diff_weeks weeks, $diff_days days, ";
print "$diff_hours hours, $diff_minutes minutes, and $diff_seconds ";
print "seconds elapsed between them.";
The two dates have 128 weeks, 6 days, 14 hours, 3 minutes,
and 45 seconds elapsed between them.
```

Обратите внимание, что разность не разделена на более крупные части (т. е. месяцы и годы), поскольку эти части имеют переменную длину и не обеспечат точного вычисления времени по полученной разности.

Обсуждение

В данном случае происходят кое-какие странные вещи, которые необходимо осмыслить. Во-первых, 1962 и 1965 годы предшествуют началу эпохи. К счастью, функция `mktime()` сбоит элегантно и для каждого года вырабатывает отрицательную метку времени. Это устраивает нас, поскольку необходимо не абсолютное значение каждой из сомнительных меток времени, а только разность между ними. До тех пор пока значения меток времени попадают в допустимый диапазон целого со знаком, их разность вычисляется правильно.

Во-вторых, настенные часы (или календарь) покажут немного другую разность во времени между этими двумя датами, поскольку они находятся по разные стороны от момента перехода на летнее время. Результат вычитания меток времени отражает правильное количество прошедшего времени, но изменение времени, воспринимаемое человеком, на час меньше. Например, какова разница между 1:30 A.M. and 4:30 A.M. апрельским воскресным утром, когда вступает в силу летнее время? Нам кажется, что 3 часа, но эти метки времени разделяет лишь 7 200 секунд, то есть два часа. Когда местные часы переводятся на один час вперед (или на час назад в октябре), то равномерный ход меток времени не принимается во внимание. В действительности прошло только два часа, хотя в результате наших манипуляций с часами это выглядит как три.

Если необходимо определить реально прошедшее время (как обычно бывает), то данный метод подходит. Если вас больше интересует разность показаний часов между двумя моментами времени, то для подсчета интервала следует опираться на юлианское представление дат, как показано в рецепте 3.6.

Чтобы сообщить пользователю о времени, прошедшем с его последней регистрации, необходимо определить разность между временем текущей регистрации и временем его последней регистрации:

```
$epoch_1 = time();
$r = mysql_query("SELECT UNIX_TIMESTAMP(last_login) AS login
                  FROM user WHERE id = $id") or die();
$ob = mysql_fetch_object($r);
$epoch_2 = $ob->login;

$diff_seconds  = $epoch_1 - $epoch_2;
$diff_weeks    = floor($diff_seconds/604800);
$diff_seconds -= $diff_weeks   * 604800;
$diff_days     = floor($diff_seconds/86400);
$diff_seconds -= $diff_days    * 86400;
$diff_hours    = floor($diff_seconds/3600);
$diff_seconds -= $diff_hours   * 3600;
$diff_minutes  = floor($diff_seconds/60);
$diff_seconds -= $diff_minutes * 60;

print "You last logged in $diff_weeks weeks, $diff_days days, ";
print "$diff_hours hours, $diff_minutes minutes, and $diff_seconds ago.";
```

См. также

Рецепт 3.6 о том, как определить разность между двумя датами с помощью юлианского представления дат; рецепт 3.10 о том, как складывать и вычитать даты; документацию по функции MySQL `UNIX_TIMESTAMP()` на *http://www.mysql.com/doc/D/a/Date_and_time_functions.html*.

3.6. Определение разности между датами юлианского календаря

Задача

Необходимо определить разность между двумя датами по показаниям часов, но отражающую действительно прошедшее время.

Решение

Функция `gregoriantojd()` позволяет получить юлианскую дату для набора частей дат, после чего можно найти разность между датами, вычтя одну юлианскую дату из другой. Затем надо преобразовать части времени в секунды и вычесть одну из другой для определения разности во времени. Если разность во времени меньше нуля, то следует уменьшить разность в датах на единицу и скорректировать разность во времени применительно к предыдущему дню. Ниже приведен соответствующий код:

```
$diff_date = gregoriantojd($date_1_mo, $date_1_dy, $date_1_yr) -
             gregoriantojd($date_2_mo, $date_2_dy, $date_2_yr);
$diff_time = $date_1_hr * 3600 + $date_1_mn * 60 + $date_1_sc -
             $date_2_hr * 3600 - $date_2_mn * 60 - $date_2_sc;
if ($diff_time < 0) {
```

```
    $diff_date--;
    $diff_time = 86400 - $diff_time;
}
```

Обсуждение

Определение разности на основе юлианского представления дат позволяет работать за пределами диапазона секунд эпохи UNIX, а также учитывать переход на летнее время.

Если компоненты двух дат находятся в массивах:

```
// 7:32:56 pm 10 мая 1965 года
list($date_1_yr, $date_1_mo, $date_1_dy, $date_1_hr, $date_1_mn, $date_1_sc)=
    array(1965, 5, 10, 19, 32, 56);
// 4:29:11 am 20 ноября 1962 года
list($date_2_yr, $date_2_mo, $date_2_dy, $date_2_hr, $date_2_mn, $date_2_sc)=
    array(1962, 11, 20, 4, 29, 11);

$diff_date = gregoriantojd($date_1_mo, $date_1_dy, $date_1_yr) -
             gregoriantojd($date_2_mo, $date_2_dy, $date_2_yr);
$diff_time = $date_1_hr * 3600 + $date_1_mn * 60 + $date_1_sc -
             $date_2_hr * 3600 - $date_2_mn * 60 - $date_2_sc;
if ($diff_time < 0) {
   $diff_date--;
   $diff_time = 86400 - $diff_time;
}
$diff_weeks = floor($diff_date/7); $diff_date -= $diff_weeks * 7;
$diff_hours = floor($diff_time/3600); $diff_time -= $diff_hours * 3600;
$diff_minutes = floor($diff_time/60); $diff_time -= $diff_minutes * 60;

print "The two dates have $diff_weeks weeks, $diff_date days, ";
print "$diff_hours hours, $diff_minutes minutes, and $diff_time ";
print "seconds between them.";
The two dates have 128 weeks, 6 days, 15 hours, 3 minutes,
and 45 seconds between them.
```

Это способ получения разности во времени по показаниям часов, поэтому результат на час превышает результат рецепта 3.5. 10 мая находится в пределах DST, а 11 ноября находится в пределах стандартного времени.

Функция `gregoriantojd()` — часть модуля `calendar` и поэтому доступна, только если PHP собран с его поддержкой (версия для Windows имеет встроенную поддержку данного расширения).

См. также

Рецепт 3.5 о том, как определить разность между двумя датами в виде прошедшего времени; рецепт 3.10 о том, как складывать и вычитать даты; документацию по функции `gregoriantojd()` на *http://www.php.net/gregoriantojd*; обзор по юлианской системе представления дат на *http://tycho.usno.navy.mil/mjd.html*.

3.7. Определение дня недели, месяца, года или номера недели в году

Задача

Необходимо узнать день или неделю года, день недели или день месяца. Например, требуется выводить на печать специальное сообщение каждый понедельник или в первый понедельник каждого месяца.

Решение

Надо передать соответствующие аргументы функции `date()` или `strftime()`:

```
print strftime("Today is day %d of the month and %j of the year.");
print 'Today is day '.date('d').' of the month and '.date('z').' of the year.';
```

Обсуждение

Эти две функции, `date()` и `strftime()`, ведут себя по-разному. Отсчет дней года начинается с 0 в функции `date()`, но с 1 в функции `strftime()`. В табл. 3.4 приведены все символы форматирования номеров дня и недели, поддерживаемые функциями `date()` и `strftime()`.

Таблица 3.4. Символы форматирования номеров дня и недели

Тип	strftime()	date()	Описание	Диапазон
День	%d	d	День месяца, численное значение	01–31
День	%j	z	День года, численное значение	001–366 для strftime(); 0–365 для date()
День	%u		День недели, численное значение (Понедельнику соответствует 1)	1–7
День	%w	w	День недели, численное значение (Воскресенью соответствует 0)	0–6
День	%W		ISO 8601 день недели, численное значение (Первый день недели – понедельник)	0–6
Неделя	%U		Номер недели в году; численное значение; первое воскресенье – это первый день первой недели	00–53
Неделя	%V	W	ISO 8601:1988 номер недели в году; численное значение; неделя 1 – это первая неделя, которая имеет как минимум четыре дня в текущем году; понедельник – это первый день недели	01–53

Например, можно вывести на печать что-нибудь только в понедельник, задав символ форматирования `w` в функции `date()` или строку `%w` в функции `strftime()`:

```
if (1 == date('w')) {
    print "Welcome to the beginning of your work week.";
}

if (1 == strftime('%w')) {
    print "Only 4 more days until the weekend!";
}
```

Существуют различные способы определения номеров недель или номеров дней в неделе, поэтому будьте внимательнее при выборе соответствующего способа. Стандарт ISO (ISO 8601) устанавливает, что недели начинаются в понедельник, а дни недели нумеруются от 1 (понедельник) до 7 (воскресенье). Неделя 1 – это первая неделя года, включающая четверг этого года. Это значит, что первая неделя в году – это неделя, в которой большинство дней принадлежат этому году. Номера недель лежат в диапазоне от 01 до 53.

Другой стандарт устанавливает диапазон номеров недель от 00 до 53, при этом дни 53-й недели года могут пересечься с днями 00-й недели следующего года.

До тех пор пока вы находитесь в рамках своей программы, можно не беспокоиться о каких-либо проблемах, но будьте осторожны при взаимодействии с другими PHP-программами или базами данных. Например, MySQL-функция `DAYOFWEEK()` считает воскресенье первым днем недели, но нумерует дни с 1 до 7, что является стандартом ODBC. Однако функция `WEEKDAY()` первым днем недели считает понедельник, а дни нумерует от 0 до 6. Функция `WEEK()` позволяет выбрать первый день недели между воскресеньем и понедельником, но она не совместима со стандартом ISO.

См. также

Документацию по функции `date()` на *http://www.php.net/date* и по функции `strftime()` на *http://www.php.net/ strftime*; документацию по функциям MySQL `DAYOFWEEK()`, `WEEKDAY()` и `WEEK()` на *http://www.mysql.com/doc/D/a/Date_and_time_functions.html*.

3.8. Проверка корректности даты

Задача

Необходимо проверить корректность даты. Например, убедиться, что пользователь не ввел определенную дату рождения, например 30 февраля 1962 года.

Решение

Проверяется с помощью функции `checkdate()`:

```
$valid = checkdate($month,$day,$year);
```

Обсуждение

Функция `checkdate()` возвращает `true`, если переменная `$month` имеет значение между 1 и 12, переменная `$year` имеет значение между 1 и 32767, а переменная `$day` находится в интервале от 1 до корректного максимального числа дней для переменных `$month` и `$year`. Високосные года корректно обрабатываются с помощью функции `checkdate()`, при этом даты представляются с использованием Григорианского календаря.

Функция `checkdate()` поддерживает весьма широкий диапазон допустимых лет, поэтому необходима дополнительная проверка пользовательского ввода, если, например, ожидается ввод даты рождения. Книга мировых рекордов Гиннеса утверждает, что возраст старейшего жителя когда-то достиг 122 лет. Убедиться, что дата рождения пользователя находится в пределах между 18 и 122 годами, можно посредством функции `pc_checkbirthdate()`, приведенной в примере 3.1.

Пример 3.1. pc_checkbirthdate( )

```
function pc_checkbirthdate($month,$day,$year) {
    $min_age = 18;
    $max_age = 122;

    if (! checkdate($month,$day,$year)) {
        return false;
    }

    list($this_year,$this_month,$this_day) = explode(',',date('Y,m,d'));

    $min_year = $this_year - $max_age;
    $max_year = $this_year - $min_age;

    print "$min_year,$max_year,$month,$day,$year\n";

    if (($year > $min_year) && ($year < $max_year)) {
        return true;
    } elseif (($year == $max_year) &&
              (($month < $this_month) ||
               (($month == $this_month) && ($day <= $this_day)))) {
        return true;
    } elseif (($year == $min_year) &&
              (($month > $this_month) ||
               (($month == $this_month && ($day > $this_day))))) {
        return true;
    } else {
        return false;
    }
}
```

Далее приведено несколько способов применения:

```
// проверка даты 3 декабря 1974 года
if (pc_checkbirthdate(12,3,1974)) {
    print "You may use this web site.";
} else {
    print "You are too young to proceed.";
    exit();
}
```

В этой функции сначала вызывается функция `checkdate()`, проверяющая корректность значений переменных `$month`, `$day` и `$year`. Затем выполняются различные сравнения, позволяющие удостовериться, что введенные даты находятся в диапазоне, установленном переменными `$min_age` и `$max_age`.

Если переменная `$year` находится в диапазоне от `$min_year` до `$max_year`, не включая границы, то дата определенно находится внутри диапазона, и функция возвращает `true`. Если нет, то выполняются некоторые дополнительные проверки. Если значение переменной `$year` равно `$max_year` (например в 2002 году переменная `$year` равна 1984), то значение переменной `$month` должно быть меньше номера текущего месяца. Если значение переменной `$month` равно номеру текущего месяца, то значение переменной `$day` должно быть меньше или равно номеру текущего дня. Если значение переменной `$year` равно `$min_year` (например в 2002 году переменная `$year` равна 1880), то значение переменной `$month` должно превышать номер текущего месяца. Если значение переменной `$month` равно номеру текущего месяца, то значение переменной `$day` должно быть больше номера текущего дня. Если не выполняется ни одно из этих условий, то введенная дата находится вне соответствующего диапазона, и функция возвращает `false`.

Функция возвращает `true`, если введенная дата отстает от текущей ровно на `$min_age` лет, но возвращает `false`, если введенная дата опережает текущую ровно на `$max_age` лет. Другими словами, это 18-й день рождения, а не 123-й.

См. также

Документацию по функции `checkdate()` на *http://www.php.net/checkdate*; информацию о долгожителе, рекордсмене из книги рекордов Гиннеса на *http://www.guinnessworldrecords.com* (найдите раздел «The Human Body», «Age and Youth», а затем «Oldest Woman Ever»).

3.9. Выделение дат и времен из строк

Задача

Необходимо извлечь из строки дату или время в формате, пригодном для вычислений. Например, конвертировать представление даты, такое как «last Thursday» (последний четверг) в метку времени UNIX.

Решение

Проще всего анализировать строки даты или времени с помощью функции strtotime(), которая превращает множество понятных человеку строк даты и времени в метку времени UNIX:

```
$a = strtotime('march 10'); // по умолчанию в текущий год
```

Обсуждение

Грамматика функции strtotime() и сложна и обширна, поэтому лучший способ освоиться с ней состоит в том, чтобы попробовать множество различных представлений времени. Те, кому интересны ее секреты, могут обратититься к файлу *ext/standard/parsedate.y* из дистрибутива PHP.

Функция strtotime() понимает слова, описывающие текущее время:

```
$a = strtotime('now');
print strftime('%c',$a);
$a = strtotime('today');
print strftime('%c',$a);
Mon Aug 12 20:35:10 2002
Mon Aug 12 20:35:10 2002
```

Она понимает различные способы представления времени и даты:[1]

```
$a = strtotime('5/12/1994');
print strftime('%c',$a);
$a = strtotime('12 may 1994');
print strftime('%c',$a);
Thu May 12 00:00:00 1994
Thu May 12 00:00:00 1994
```

Она понимает относительные время и дату:

```
$a = strtotime('last thursday');   // 12 августа 2002 года
print strftime('%c',$a);
$a = strtotime('2001-07-12 2pm + 1 month');
print strftime('%c',$a);
Thu Aug  8 00:00:00 2002
Mon Aug 12 14:00:00 2002
```

Она понимает часовые пояса. Когда следующий код запускается на компьтере, находящемся в зоне восточного поясного летнего времени (Eastern Daylight Time, EDT), он выводит то же самое время:

```
$a = strtotime('2002-07-12 2pm edt + 1 month');
print strftime('%c',$a);
Mon Aug 12 14:00:00 2002
```

[1] Формат представления даты «день/месяц/год», принятый в России, не поддерживается. – *Примеч. науч. ред.*

Однако если код, приведенный ниже, запустить на компьютере, находящемся в поясе EDT, то он выведет время в часовом поясе EDT (16:00), когда в зоне горного летнего времени (Mountain Daylight Time, MDT), расположенной на 2 часа ближе к Гринвичу, чем пояс EDT, наступает два часа пополудни (14:00):[1]

```
$a = strtotime('2002-07-12 2pm mdt + 1 month');
print strftime('%c',$a);
Mon Aug 12 16:00:00 2002
```

Если дата и время, которые требуется выделить из строки, представлены в известном заранее формате, то можно не вызывать функцию `strtotime()`, а сконструировать регулярное выражение, выделяющее необходимые части даты и времени. Ниже показано, как разобрать даты формата «YYYY-MM-DD HH:MM:SS», такие как поле `DATETIME` в MySQL:

```
$date = '1974-12-03 05:12:56';
preg_match('/(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})/
',$date,$date_parts);
```

Этот фрагмент помещает год, месяц, день, час, минуту и секунду в переменные от `$date_parts[1]` до `$date_parts[6]`. (Функция `preg_match()` помещает все выделенное выражение в переменную `$date_parts[0]`.)

Регулярные выражения позволяют извлечь дату и время из большей строки, которая может также содержать и другую информацию (из ввода пользователя или из файла), но если известно расположение даты в разбираемой строке, то вызов функции `substr()` может даже ускорить разбор строки:

```
$date_parts[0] = substr($date,0,4);
$date_parts[1] = substr($date,5,2);
$date_parts[2] = substr($date,8,2);
$date_parts[3] = substr($date,11,2);
$date_parts[4] = substr($date,14,2);
$date_parts[5] = substr($date,17,2);
```

Можно также использовать функцию `split()`;

```
$ar = split('[- :]',$date);
print_r($ar);
Array
(
    [0] => 1974
    [1] => 12
    [2] => 03
    [3] => 05
    [4] => 12
    [5] => 56
)
```

[1] Таблицу часовых поясов США можно найти на странице *http://www.space-archive.info/utc.htm*, а описание всех часовых поясов – по адресу *http://en.wikipedia.org/wiki/Time_zone#UTC.2B6_F*. – *Примеч. ред.*

Будьте осторожны: PHP выполняет преобразование между числами и строками без какого-либо предупреждения, но числа, начинающиеся с 0, считаются восьмеричными (по основанию 8). Поэтому 03 и 05 – это 3 и 5; но 08 и 09 – это *не* 8 и 9.

Функции `preg_match()` и `strtotime()` имеют одинаковую эффективность при анализе формата даты, такого как «YYYY-MM-DD HH:MM:SS», но функция `ereg()` работает почти в четыре раза медленнее, чем другие. Для отделения части строки даты функция `preg_match()` наиболее удобна, но функция `strtotime()` очевидно имеет намного большую гибкость.

См. также

Документацию по функции `strtotime()` на *http://www.php.net/strtotime*; грамматику функции `strtotime()`, доступную на *http://cvs.php.net/cvs.php/php4/ext/standard/parsedate.y*.

3.10. Сложение и вычитание дат

Задача

Необходимо добавить или вычесть интервал из даты.

Решение

В зависимости от способа представления даты и интервала, следует применять функцию `strtotime()` или некоторые простые арифметические функции.

Если дата и интервал представлены в соответствующем формате, то проще обратиться к функции `strtotime()`:

```
$birthday = 'March 10, 1975';
$whoopee_made = strtotime("$birthday - 9 months ago");
```

Если дата представлена в виде метки времени UNIX, а интервал можно выразить в секундах, то надо вычесть интервал из метки времени:

```
$birthday = 163727100;
$gestation = 36 * 7 * 86400; // 36 weeks
$whoopee_made = $birthday - $gestation;
```

Обсуждение

Функцию `strtotime()` удобно применять с интервалами переменной длины, такими как месяцы. Если нельзя использовать эту функцию, то можно преобразовать дату в метку времени и добавить или вычесть интервал в секундах. Это удобнее всего для интервалов с фиксированным временем, таких как дни или недели:

```
$now = time();
$next_week = $now + 7 * 86400;
```

Однако данный способ может привести к трудностям, если границы интервала находятся по разные стороны от момента перехода на летнее время. В этом случае длина одного из дней не будет равна 86 400 секундам, а составит либо 82 800, либо 90 000 секунд, в зависимости от сезона. Если приложение работает исключительно с UTC, то об этом можно не беспокоиться. Но если необходимо учитывать местное время, то избежать трудностей при подсчете дней поможет юлианское представление дат. Преобразования между метками времени и юлианскими датами обечпечивают функции `unixtojd()` и `jdtounix()`:

```
$now = time();
$today = unixtojd($now);
$next_week = jdtounix($today + 7);
// don't forget to add back hours, minutes, and seconds
$next_week += 3600 * date('H',$now) + 60 * date('i',$now) + date('s',$now);
```

Функции `unixtojd()` и `jdtounix()` входят в модуль `calendar` и поэтому доступны, только если PHP собран с его поддержкой (версия для Windows имеет встроенную поддержку данного расширения).

См. также

Рецепт 3.5 о том, как определять разность между двумя датами в виде прошедшего времени; рецепт 3.6 о том, как определять разность между двумя датами в юлианском представлении дат; документацию по функции `strtotime()` на *http://www.php.net/strtotime*, по функции `unixtojd()` на *http://www.php.net/unixtojd* и по функции `jdtounix()` на *http://www.php.net/jdtounix*.

3.11. Учет часовых поясов при определении времени

Задача

Необходимо вычислить время в различных часовых поясах. Например, надо сообщить пользователю информацию, привязанную к его местному времени, а не к местному времени вашего сервера.

Решение

В случае простых вычислений можно непосредственно добавить или вычесть разность между двумя часовыми поясами:

```
// если локальное время - это зона EST
$time_parts = localtime();
// Калифорния (PST) на три часа раньше
$california_time_parts = localtime(time() - 3 * 3600);
```

В UNIX-системах, если не известно смещение между временными зонами, достаточно установить значение переменной окружения равным целевому часовому поясу:

```
putenv('TZ=PST8PDT');
$california_time_parts = localtime();
```

Обсуждение

Перед тем как начать обходить все углы и закоулки часовых поясов, мы хотим передать заявление, которое военно-морская обсерватория США представляет на *http://tycho.usno.navy.mil/tzones.html*. А именно, что официальная информация о часовых поясах мира в некотором роде непостоянна, «поскольку нации суверенны и могут изменять и изменяют свои системы хранения времени так, как они считают нужным». Поэтому, помня о превратностях международных отношений, которые властвуют над нами, мы прибегаем к нескольким способам, позволяющим правильно работать с часовыми поясами.

Для относительно простой обработки смещений между часовыми поясами в программе организуется массив, в котором хранятся различные смещения относительно пояса UTC. Определив часовой пояс пользователя, просто прибавьте это смещение к соответствующему UTC времени, и функции, которые выводят на печать UTC время (т. е. `gmdate()`, `gmstrftime()`), смогут напечатать соответствующим образом скорректированное время.

```
// Определяем текущее время
$now = time();

// Калифорния на 8 часов отстает от зоны UTC
$now += $pc_timezones['PST'];

// Используйте функции gmdate() или gmstrftime()
// для вывода Калифорнийского времени
print gmstrftime('%c',$now);
```

В вышеприведенном коде смещения относительно зоны UTC хранятся в массиве `$pc_timezones`:

```
// Из Perl Time::Timezone
$pc_timezones = array(
  'GMT'  =>   0,           // Среднее время по гринвичскому меридиану
  'UTC'  =>   0,           // Универсальное (Скоординированное) время
  'WET'  =>   0,           // Западноевропейское время
  'WAT'  =>  -1*3600,      // Западноафриканское время
  'AT'   =>  -2*3600,      // Время Азорских островов
  'NFT'  =>  -3*3600-1800, // Время Ньюфаундленда
  'AST'  =>  -4*3600,      // Стандартное Атлантическое время
  'EST'  =>  -5*3600,      // Восточное стандартное время
  'CST'  =>  -6*3600,      // Центральное стандартное время
  'MST'  =>  -7*3600,      // Стандартное Горное время
  'PST'  =>  -8*3600,      // Стандартное Тихоокеанское время
```

```
    'YST'  =>  -9*3600,      // Стандартное Юконское время
    'HST'  => -10*3600,      // Стандартное Гавайское время
    'CAT'  => -10*3600,      // Время Центральной Аляски
    'AHST' => -10*3600,      // Стандартное время Аляска-Гавайи
    'NT'   => -11*3600,      // Время города Ном
    'IDLW' => -12*3600,      // К западу от международной линии смены дат
    'CET'  =>  +1*3600,      // Центральноевропейское время
    'MET'  =>  +1*3600,      // Среднеевропейское время
    'MEWT' =>  +1*3600,      // Зимнее Среднеевропейское время
    'SWT'  =>  +1*3600,      // Зимнее Шведское время
    'FWT'  =>  +1*3600,      // Зимнее Французское время
    'EET'  =>  +2*3600,      // Восточноевропейское время, СССР Зона 1
    'BT'   =>  +3*3600,      // Багдадское время, СССР Зона 2
    'IT'   =>  +3*3600+1800, // Иранское время
    'ZP4'  =>  +4*3600,      // СССР Зона 3
    'ZP5'  =>  +5*3600,      // СССР Зона 4
    'IST'  =>  +5*3600+1800, // Стандартное Индийское время
    'ZP6'  =>  +6*3600,      // СССР Зона 5
    'SST'  =>  +7*3600,      // Южносуматранское время, СССР Зона 6
    'WAST' =>  +7*3600,      // Стандартное Западноавстралийское время
    'JT'   =>  +7*3600+1800, // Время острова Ява
    'CCT'  =>  +8*3600,      // Время Китайского побережья, СССР Зона 7
    'JST'  =>  +9*3600,      // Стандартное Японское, СССР Зона 8
    'CAST' =>  +9*3600+1800, // Стандартное Центральноавстралийское время
    'EAST' => +10*3600,      // Стандартное Восточноавстралийское время
    'GST'  => +10*3600,      // Стандартное Гуамское время, СССР Зона 9
    'NZT'  => +12*3600,      // Новозеландское время
    'NZST' => +12*3600,      // Стандартное Новозеландское время
    'IDLE' => +12*3600       // К востоку от международной линии смены дат
);
```

В UNIX-системах преобразования можно выполнять при помощи библиотеки *zoneinfo*. Это позволяет сделать код более компактным, а обработку DST более прозрачной, как показано в рецепте 3.12.

Чтобы получить преимущества библиотеки *zoneinfo* в PHP, все вычисления с международными датами надо выполнять на основе меток времени UNIX. Последние следует генерировать из частей времени с помощью функции `pc_mktime()`, показанной в примере 3.2.

Пример 3.2. pc_mktime( )

```
function pc_mktime($tz,$hr,$min,$sec,$mon,$day,$yr) {
    putenv("TZ=$tz");
    $a = mktime($hr,$min,$sec,$mon,$day,$yr);
    putenv('TZ=EST5EDT');   // замена пояса EST5EDT на часовой пояс
                               вашего сервера!
    return $a;
}
```

Вызов функции `putenv()` до вызова `mktime()` позволяет обмануть системную функцию `mktime()`, заставляя ее думать, что она находится в дру-

гом часовом поясе. После вызова функции mktime() необходимо восстановить истинный часовой пояс. На восточном побережье Соединеных Штатов это пояс EST5EDT. Замените его соответствующим значением географического положения вашего компьютера (см. далее).

Части времени преобразуются в метки времени с помощью функции pc_mktime(). Ее двойником, который превращает метки времени в форматированную строку времени и части времени, является функция pc_strftime(), показанная в примере 3.3.

Пример 3.3. pc_strftime( )

```
function pc_strftime($tz,$format,$timestamp) {
    putenv("TZ=$tz");
    $a = strftime($format,$timestamp);
    putenv('TZ=EST5EDT');   // замена пояса EST5EDT на часовой пояс
                               вашего сервера!
    return $a;
}
```

В этом примере применен тот же способ обмана системной функции, к которому прибегает функция pc_mktime() для получения правильного результата от функции strftime().

Существенным в этих функциях является то, что не надо беспокоиться о смещениях различных временных поясов относительно UTC независимо от того, действует ли переход на летнее время или какие-либо другие особенности часовых поясов. Достаточно установить соответствующий часовой пояс, а системная библиотека сделает все остальное.

Учтите, что значение переменной $tz в обеих функциях должно быть именем не часового пояса, а зоны *zoneinfo*. Пояса *zoneinfo* имеют больше особенностей, чем часовые пояса, так как они соответствуют определенным местам. Табл. 3.5 показывает соответствия между определенными поясами *zoneinfo* и смещениями относительно зоны UTC. Последняя колонка показывает, происходит ли в данном поясе переход на летнее время.

Таблица 3.5. Зоны zoneinfo

UTC смещение (часы)	UTC смещение (секунды)	зона zoneinfo	DST?
–12	–43200	Etc/GMT+12	Нет
–11	–39600	Тихий океан/Остров Мидуэй	Нет
–10	–36000	США/Алеутские острова	Да
–10	–36000	Тихий океан/Гонолулу	Нет
–9	–32400	Америка/Анкоридж	Да
–9	–32400	Etc/GMT+9	Нет
–8	–28800	PST8PDT	Да

Таблица 3.5 (продолжение)

UTC смещение (часы)	UTC смещение (секунды)	зона zoneinfo	DST?
–8	–28800	Америка/Даусон-Крик	Нет
–7	–25200	MST7MDT	Да
–7	–25200	MST	Нет
–6	–21600	CST6CDT	Да
–6	–21600	Канада/Река Саскачеван	Нет
–5	–18000	EST5EDT	Да
–5	–18000	EST	Нет
–4	–14400	Америка/Галифакс	Да
–4	–14400	Америка/Пуэрто-Рико	Нет
–3,5	–12600	Америка/Сэйнт-Джонс	Да
–3	–10800	Америка/Буэнос-Айрес	Нет
0	0	Европа/Лондон	Да
0	0	GMT	Нет
1	3600	CET	Да
1	3600	GMT-1	Нет
2	7200	EET	Нет
2	7200	GMT-2	Нет
3	10800	Азия/Багдад	Да
3	10800	GMT-3	Нет
3,5	12600	Азия/Тегеран	Да
4	14400	Азия/Дубай	Нет
4	14400	Азия/Баку	Да
4,5	16200	Азия/Кабул	Нет
5	18000	Азия/Ташкент	Нет
5,5	19800	Азия/Калькутта	Нет
5,75	20700	Азия/Катманду	Нет
6	21600	Азия/Новосибирск	Да
6	21600	Etc/GMT-6	Нет
6,5	23400	Азия/Рангун	Нет
7	25200	Азия/Джакарта	Нет
8	28800	Гонконг	Нет

UTC смещение (часы)	UTC смещение (секунды)	зона zoneinfo	DST?
9	32400	Япония	Нет
9,5	34200	Австралия/Дарвин	Нет
10	36000	Австралия/Сидней	Да
10	36000	Тихий океан/Остров Гуам	Нет
12	43200	Etc/GMT-13	Нет
12	43200	Тихий океан/Окленд	Да

В разных странах переход на летнее и зимнее время не происходит в один и тот же день. Чтобы вычислить время с учетом перехода на летнее время в соответствии с географическим положением пункта, выберите пояс *zoneinfo*, который ближе всего находится к данному пункту.

См. также

Рецепт 3.12 о том, как работать с DST; документацию по функции `putenv()` на *http://www.php.net/ putenv*, по функции `localtime()` на *http://www.php.net/localtime*, по функции `gmdate()` на *http://www.php.net/gmdate* и по функции `gmstrftime()` на *http://www.php.net/gmstrftime*; имена часовых поясов *zoneinfo*, а также широту и долготу сотен населенных пунктов мира – на *ftp://elsie.nci.nih.gov/pub/tzdata2002c.tar.gz*; ссылки на историческую и техническую информацию о часовых поясах можно найти на *http://www.twinsun.com/tz/tz-link.htm*.

3.12. Учет перехода на летнее время

Задача

Необходимо обеспечить корректное определение времени с учетом перехода на летнее время.

Решение

Библиотека *zoneinfo* производит верные вычисления с учетом DST. В UNIX-системах преимущества *zoneinfo* обеспечивает функция `putenv()`:

```
putenv('TZ=MST7MDT');
print strftime('%c');
```

Если *zoneinfo* нельзя использовать, то можно изменить жестко запрограммированные смещения часовых поясов, с учетом наличия или отсутствия поддержки DST локальным часовым поясом. Функция `localtime()` позволяет определить текущий статус поддержки DST:

```
// Определение текущего времени UTC
$now = time();
```

```
// Калифорния на 8 часов отстает от зоны UTC
$now -= 8 * 3600;

// DST действует?
$ar = localtime($now,true);
if ($ar['tm_isdst']) { $now += 3600; }

// Используем функцию gmdate() или gmstrftime()
// для печати Калифорнийского времени
print gmstrftime('%c',$now);
```

Обсуждение

Замена метки времени на значение смещения часового пояса относительно зоны UTC и последующий вызов функции `gmdate()` или `gmstrftime()` для вывода на печать значений функций, соответствующих часовому поясу, – это гибкий метод, работающий в любой временной зоне, но вычисления DST не совсем точны. Для коротких интервалов времени, когда DST-статус сервера отличается от статуса целевого часового пояса, результат неверен. Например, в 3:30 A.M. EDT в первое воскресенье апреля (после перехода на летнее время) в Тихоокеанском часовом поясе время (11:30 P.M.) еще зимнее. Сервер в Восточном часовом поясе, использующий этот метод, определит, что Калифорнийское время отстает от зоны UTC на семь часов, тогда как в действительности отставание составляет восемь часов. В 6:00 A.M. EDT (3:00 A.M. PDT) и Тихоокеанское и Восточное время уже учитывают DST, поэтому вычисления снова правильные (отставание Калифорнии от зоны UTC составляет восемь часов).

См. также

Рецепт 3.11 о том, как работать с часовыми поясами; документацию по функции `putenv()` на *http://www.php.net/putenv*, по функции `localtime()` на *http://www.php.net/localtime*, по функции `gmdate()` на *http://www.php.net/gmdate* и по функции `gmstrftime()` на *http://www.php.net/gmstrftime*; переход на летнее время (DST) подробно рассмотрен на *http://webexhibits.org/daylightsaving/*.

3.13. Выработка высокоточного времени

Задача

Необходимо измерять время с более чем секундной точностью, например для того, чтобы сгенерировать уникальный идентификатор.

Решение

Это можно сделать при помощи функции `microtime()`:

```
list($microseconds,$seconds) = explode(' ',microtime());
```

Обсуждение

Функция `microtime()` возвращает строку, содержащую дробную часть времени, прошедшего с начала эпохи, с точностью до микросекунды. Например, возвращенное значение `0.41644100 1026683258` означает, что с начала эпохи прошло **1026683258,41644100** секунд. Вместо числа с двойной точностью возвращается строка, поскольку число с двойной точностью не настолько велико, чтобы хранить полное значение с микросекундной точностью.

Значение времени, содержащее миросекунды, удобно для генерирования уникальных идентификаторов. Объединение его с идентификатором текущего процесса гарантирует получение уникального значения, поскольку процесс не может генерировать более одного идентификатора за одну микросекунду:

```
list($microseconds,$seconds) = explode(' ',microtime());
$id = $seconds.$microseconds.getmypid();
```

Однако этот метод не так надежен в многопотоковых системах, где существует небольшая (но отличная от нуля) вероятность, что два потока одного процесса одновременно вызовут функцию `microtime()`.

См. также

Документацию по функции `microtime()` на *http://www.php.net/microtime*.

3.14. Получение интервалов времени

Задача

Необходимо знать все дни недели или месяца. Например, требуется вывести на печать список встреч на протяжении недели.

Решение

Определите начальную дату с помощью функции `time()` и `strftime()`. Если интервал имеет фиксированную длину, то можно выполнить цикл по дням этого интервала. Если нет, то необходимо проверять каждую подпоследовательность дней на принадлежность к требуемому диапазону.

Например, в неделе семь дней, поэтому для выработки дней текущей недели можно выполнить фиксированный цикл:

```
// генерация интервала времени для этой недели
$now = time();

// Если это время до 3 AM, увеличьте $now на 1, чтобы не беспокоиться
// о DST при движении обратно к началу недели.
if (3 < strftime('%H', $now)) { $now += 7200; }
```

```
// Какой сегодня день недели?
$today = strftime('%w', $now);

// Сколько дней назад началась неделя?
$start_day = $now - (86400 * $today);

// Вывод каждого дня недели
for ($i = 0; $i < 7; $i++) {
  print strftime('%c',$start_day + 86400 * $i);
}
```

Обсуждение

Отдельный месяц или год может иметь различное число дней, поэтому необходимо определить конец временного диапазона, учитывая особенности данного месяца или года. Чтобы выполнить цикл по всем дням месяца, определите метку времени для первого дня этого месяца и для первого дня следующего месяца. Переменная цикла `$day` содержит день, увеличивающийся на определенное время (86 400 секунд), пока он не превысит метку времени начала следующего месяца:

```
// Генерирует временной интервал для месяца
$now = time();

// Если это до 3 AM, увеличьте $now на 1 так, чтобы не беспокоиться
// о DST при движении обратно к началу недели.
if (3 < strftime('%H', $now)) { $now += 7200; }

// Какой сегодня день недели?
$this_month = strftime('%m',$now);

// Метка времени полуночи первого дня этого месяца
$day = mktime(0,0,0,$this_month,1);
// Метка времени полуночи первого дня следующего месяца
$month_end = mktime(0,0,0,$this_month+1,1);

while ($day < $month_end) {
  print strftime('%c',$day);
  $day += 86400;
}
```

См. также

Документацию по функции `time()` на *http://www.php.net/time* и по функции `strftime()` на *http://www.php.net/strftime*.

3.15. Работа с негригорианскими календарями

Задача

Необходимо работать с негригорианским календарем, таким как Юлианский, Иудейский или Французский Республиканский.

Решение

Модуль PHP «календарь» обеспечивает функции преобразования для работы с Юлианским календарем, так же как и с Французским Республиканским и Иудейским календарями. Для работы с этими функциями необходимо, чтобы PHP был собран с поддержкой этого модуля.

Эти функции используют юлианский счет дней (а это не то же самое, что Юлианский календарь) как промежуточный формат обмена данными между ними.

Две функции, `jdtogregorian()` и `gregoriantojd()`, выполняют преобразования между юлианскими днями и теми же датами Григорианского календаря:

```
$jd = gregoriantojd(3,9,1876);      // 9 марта 1876 года; $jd = 2406323

$gregorian = jdtogregorian($jd);    // $gregorian = 3/9/1876
```

Допустимым диапазоном Григорианского календаря являются значения от 4714 BCE (до рождества Христова) до 9999 CE (после рождества Христова).

Обсуждение

Преобразования между юлианским представлением дат и Юлианским календарем выполняются при помощи функции `jdtojulian()` и `juliantojd()`:

```
// 29 февраля 1900 года (негригорианский високосный год)
$jd = juliantojd(2,29,1900);      // $jd = 2415092
$julian = jdtojulian($jd);        // $julian = 2/29/1900
$gregorian = jdtogregorian($jd);  // $gregorian = 3/13/1900
```

Допустимым диапазоном для Юлианского календаря являются значения от 4713 BCE до 9999 CE, но так как он был создан в 46 году до рождества Христова, то вы рискуете вызвать недовольство поклонников Юлианского календаря, если будете использовать его для дат до его создания.

Преобразования между юлианским представлением дат и Французским Республиканским календарем выполняется посредством функции `jdtofrench()` и `frenchtojd()`:

```
$jd = frenchtojd(8,13,11);        // 13 floreal XI; $jd = 2379714
$french = jdtofrench($jd);        // $french = 8/13/11
$gregorian = jdtofregorian($jd);  // $gregorian = 5/3/1803;
                                     дата продажи Луизианы США.
```

Для Французского Республиканского календаря допустимыми являются даты с сентября 1792 года до сентября 1806 года, что представляет собой небольшой интервал времени, но, поскольку календарем пользовались с 1793 года до января 1806 года, этого вполне достаточно.

Для преобразования между юлианским представлением дат и Иудейским календарем применяются функции `jdtojewish()` и `jewishtojd( )`:

```
$jd = JewishToJD(6,14,5761);       // Adar 14, 5761; $jd = 2451978
$jewish = JDToJewish($jd);         // $jewish = 6/14/5761
$gregorian = JDToGregorian($jd);   // $gregorian = 3/9/2001
```

Допустимый диапазон для Иудейского календаря начинается с 3761 ВСЕ (первый год по Иудейскому календарю).

См. также

Документацию по функциям календаря на *http://www.php.net/calendar*; история Григорианского календаря изложена на странице *http://scienceworld.wolfram.com/astronomy/ GregorianCalendar.html*.

3.16. Программа: Календарь

Функция `pc_calendar()`, показанная в примере 3.4, выводит календарь на месяц, подобно UNIX-функции *cal*. Пример:

```
// печать календаря для текущего месяца
list($month,$year) = explode(',',date('m,Y'));
pc_calendar($month,$year);
```

Функция `pc_calendar()` выводит таблицу, содержащую календарь на месяц. Календарь содержит ссылки на предыдущий и последующий месяцы и выделяет текущий день.

Пример 3.4. pc_calendar( )

```
<?php
function pc_calendar($month,$year,$opts = '') {
    // установка опций по умолчанию //
    if (! is_array($opts)) { $opts = array(); }
    if (! isset($opts['today_color'])) { $opts['today_color'] = '#FFFF00'; }
    if (! isset($opts['month_link'])) {
        $opts['month_link'] =
            '<a href="'.$_SERVER['PHP_SELF'].'?month=%d&year=%d">%s</a>';
    }

    list($this_month,$this_year,$this_day) = split(',',strftime('%m,%Y,%d'));
    $day_highlight = (($this_month == $month) && ($this_year == $year));

    list($prev_month,$prev_year) =
        split(',',strftime('%m,%Y',mktime(0,0,0,$month-1,1,$year)));
    $prev_month_link = sprintf($opts['month_link'],
                               $prev_month,$prev_year,'&lt;');

    list($next_month,$next_year) =
        split(',',strftime('%m,%Y',mktime(0,0,0,$month+1,1,$year)));
    $next_month_link = sprintf($opts['month_link'],
                               $next_month,$next_year,'&gt;');
```

```
?>
<table border="0" cellspacing="0" cellpadding="2" align="center">
        <tr>
                <td align="left">
                        <?php print $prev_month_link ?>
                </td>
                <td colspan="5" align="center">
               <?php print strftime('%B %Y',mktime(0,0,0,$month,1,$year)); ?>
                </td>
                <td align="right">
                        <?php print $next_month_link ?>
                </td>
        </tr>
<?php
    $totaldays = date('t',mktime(0,0,0,$month,1,$year));

    // выводим дни недели
    print '<tr>';
    $weekdays = array('Su','Mo','Tu','We','Th','Fr','Sa');
    while (list($k,$v) = each($weekdays)) {
        print '<td align="center">'.$v.'</td>';
    }
    print '</tr><tr>';
    // выравниваем первый день месяца по соответствующему дню недели
    $day_offset = date("w",mktime(0, 0, 0, $month, 1, $year));
    if ($day_offset > 0) {
        for ($i = 0; $i < $day_offset; $i++) { print '<td> </td>'; }
    }
    $yesterday = time() - 86400;

    // выводим дни
    for ($day = 1; $day <= $totaldays; $day++) {
        $day_secs = mktime(0,0,0,$month,$day,$year);
        if ($day_secs >= $yesterday) {
            if ($day_highlight && ($day == $this_day)) {
                print sprintf('<td align="center" bgcolor="%s">%d</td>',
                              $opts['today_color'],$day);
            } else {
                print sprintf('<td align="center">%d</td>',$day);
            }
        } else {
            print sprintf('<td align="center">%d</td>',$day);
        }
        $day_offset++;

        // начинаем новую строку каждую неделю //
        if ($day_offset == 7) {
            $day_offset = 0;
            print "</tr>\n";
            if ($day < $totaldays) { print '<tr>'; }
        }
    }
```

```
        // заполнение последней недели пробелами //
        if ($day_offset > 0) { $day_offset = 7 - $day_offset; }
        if ($day_offset > 0) {
            for ($i = 0; $i < $day_offset; $i++) { print '<td> </td>'; }
        }
        print '</tr></table>';
    }
    ?>
```

Функция pc_calendar() начинает работу с проверки параметров, переданных ей в переменной $opts. Цвет, которым выделяется текущий день, можно передать в виде RGB-значения через переменную $opts['today_color']. Значение по умолчанию равно #FFFF00, это ярко-желтый цвет. Кроме того, чтобы изменить вид вывода ссылок на предыдущий и последующий месяцы, можно передать строку форматирования в стиле функции printf() через переменную $opts['month_link'].

Затем функция устанавливает значение переменной $day_highlight в true, если месяц и год календаря совпадают с текущими месяцем и годом. Ссылки на предыдущий и последующий месяцы помещаются в переменные $prev_month_link и $next_month_link с помощью строки форматирования, находящейся в переменной $opts['month_link'].

После этого функция pc_calendar() выводит верхнюю часть HTML-таблицы, которая содержит календарь, и строку таблицы с сокращенными названиями дней недели. С учетом дня недели, возвращенного функцией strftime('%w'), печатаются пустые ячейки таблицы, чтобы первый день месяца был выровнен по соответствующему дню недели. Например, если первый день месяца вторник, то надо напечатать две пустые ячейки, чтобы занять места, отведенные под воскресенье и понедельник в первой строке таблицы.

После вывода этой предварительной информации функция pc_calendar() выполняет цикл по всем дням месяца. Для большинства дней она печатает обычные ячейки таблицы, а для текущего дня печатает ячейку с другим цветом фона. Когда значение переменной $day_offset достигает 7, то неделя заполнена, и надо начинать новую строку таблицы.

После вывода ячеек таблицы для каждого дня месяца добавляются пустые ячейки, чтобы заполнить до конца последнюю строку таблицы. Например, если последний день месяца четверг, то добавляются две ячейки, чтобы занять пространство, отведенное под пятницу и субботу. Наконец, таблица закрывается, и календарь полностью готов.

4

Массивы

4.0. Введение

Массивы – это списки: списки людей, списки размеров, списки книг. Массивы предназначены для сохранения группы элементов в переменной. Подобно списку на листке бумаги, элементы в массиве упорядочены. Обычно каждый новый элемент вставляется вслед за последним вхождением в массив, но точно так же, как вы можете вставить новую запись между двумя строчками, с массивом в PHP можно сделать то же самое.

Во многих языках существует только один тип массива: то, что называется массивом *с числовой индексацией* (или просто массивом). Для нахождения элемента необходимо знать его относительное положение в массиве, известное как *индекс*. Позиции элементов определяются числами: они начинаются с 0 и увеличиваются на единицу для каждого следующего элемента.

В других языках существует и другой тип массива: *ассоциативный массив*, известный также как хеш. В ассоциативном массиве в качестве индексов выступают не целые числа, а строки. Так, в массиве президентов США «Авраам Линкольн» мог бы иметь индекс 16, а в ассоциативной версии массива индекс мог бы быть «Honest». Однако в то время как в массиве с числовой индексацией порядок элементов продиктован его ключами и строго соблюдается, ассоциативный массив часто не гарантирует упорядочение ключей. Элементы добавляются в определенном порядке, но позже этот порядок никоим образом определить нельзя.

Некоторые языки поддерживают как массивы с числовой индексацией (просто массивы), так и ассоциативные. Но, как правило, массив `$presidents` и ассоциативный массив `$presidents` это не одно и то же. Для каждого типа массивов характерно специфическое поведение, и обращаться с ними надо соответственно. В PHP допускаются и массивы обоих типов, но они не работают самостоятельно.

В PHP массив с числовой индексацией *является* ассоциативным массивом, и наоборот. Какой же тогда тип они имеют в действительности? Оба и никакой. Граница между ними постоянно смещается от одного к другому. Поначалу это может дезориентировать, особенно тех, кто придерживается строгих правил, но скоро и они обнаружат, что такая гибкость – ценное свойство.

Для того чтобы присвоить множество значений массива за один раз, применяется функция `array()`:

```
$fruits = array('Apples', 'Bananas', 'Cantaloupes', 'Dates');
```

Теперь значение `$fruits[2]` равно `'Cantaloupes'`.

Функция `array()` очень удобна в случае короткого списка известных значений. Тот же самый массив можно получить так:

```
$fruits[0] = 'Apples';
$fruits[1] = 'Bananas';
$fruits[2] = 'Cantaloupes';
$fruits[3] = 'Dates';
```

и так:

```
$fruits[] = 'Apples';
$fruits[] = 'Bananas';
$fruits[] = 'Cantaloupes';
$fruits[] = 'Dates';
```

Присваивание значения элементу массива с пустым списком индексов – это сокращенная запись операции добавления нового элемента в конец массива. Так, PHP определяет длину массива `$fruits` и использует ее в качестве позиции нового значения. Это предполагает, конечно, что переменная `$fruits` не является скалярной, например 3, и не является объектом. PHP не выразит недовольства, если вы попробуете трактовать немассив как массив; однако если это будет первым случаем использования этой переменной, то PHP автоматически преобразует ее в массив и начнет отсчет индекса с 0.

Точно так же ведет себя функция `array_push()`, которая проталкивает новое значение на вершину стека массива. Однако для PHP запись вида `$foo[]` более традиционна, к тому же она быстрее. Но иногда функция `array_push()` точнее передает стековую природу выполняемых действий, особенно при совместном использовании с функцией `array_pop()`, которая возвращает последний элемент и удаляет его из массива.

До сих пор мы заносили в массив только целые числа и строки. Однако PHP позволяет присваивать элементам массива данные любого типа: логические значения, целые числа, числа с двойной точностью, строки, объекты, ресурсы, значение `NULL` и даже другие массивы. Поэтому можно извлечь массивы или объекты прямо из базы данных и поместить их в массив:

```
while ($row = mysql_fetch_row($r)) {
```

```
    $fruits[] = $row;
}

while ($obj = mysql_fetch_object($s)) {
    $vegetables[] = $obj;
}
```

Первый оператор `while` создает массив массивов; второй создает массив объектов. Смотрите рецепт 4.2, где хранение множества элементов по одному ключу описано более подробно.

Для того чтобы определить массив не с целочисленным ключом, а со строковым, можно также использовать функцию `array()`, но определив пару ключ/значение с помощью символа `=>`:

```
$fruits = array('red' => 'Apples', 'yellow' => 'Bananas',
                'beige' => 'Cantaloupes', 'brown' => 'Dates');
```

Теперь значение переменной `$fruits['beige']` равно `'Cantaloupes'`. Это краткая запись следующего фрагмента:

```
$fruits['red'] = 'Apples';
$fruits['yellow'] = 'Bananas';
$fruits['beige'] = 'Cantaloupes';
$fruits['brown'] = 'Dates';
```

Каждый массив может хранить только одно уникальное значение для каждого ключа. Присваивание:

```
$fruits['red'] = 'Strawberry';
```

перепишет значение `'Apple'`. Однако всегда можно добавить другой ключ позже:

```
$fruits['orange'] = 'Orange';
```

Чем больше программ вы пишете на PHP, тем привычнее и естественнее становятся ассоциативные массивы, а не массивы с числовой индексацией. Вместо того чтобы создавать массив с числовой индексацией, содержащий строковые значения, можно создать ассоциативный массив и поместить в него значения в качестве его ключей. При желании можно затем занести в элементы массива дополнительную информацию. При этом здесь не берут штраф за скорость, а PHP охраняет порядок. Плюс к тому облегчается поиск и изменение значения, поскольку уже известен его ключ.

Самый простой способ организации цикла по массиву и выполнения операций со всеми или некоторыми его элементами – это использование оператора `foreach`:

```
$fruits = array('red' => 'Apples', 'yellow' => 'Bananas',
                'beige' => 'Cantaloupes', 'brown' => 'Dates');

foreach ($fruits as $color => $fruit) {
    print "$fruit are $color.\n";
```

```
}
Apples are red.
Bananas are yellow.
Cantaloupes are beige.
Dates are brown.
```

При каждом прохождении цикла PHP присваивает значение следующего ключа переменной `fruit`. Когда в массиве не остается ни одного элемента, цикл заканчивается.

Функция `list()` позволяет разбивать массив на отдельные переменные:

```
$fruits = array('Apples', 'Bananas', 'Cantaloupes', 'Dates');

list($red, $yellow, $beige, $brown) = $fruits;
```

4.1. Определение массива с ненулевым начальным индексом

Задача

Необходимо присвоить множество значений элементам массива за один раз, но при этом первый индекс не должен быть равен 0.

Решение

Прикажите функции `array()` использовать другой индекс, применив синтаксис `=>`:

```
$presidents = array(1 => 'Washington', 'Adams', 'Jefferson', 'Madison');
```

Обсуждение

Массивы в PHP, как и в большинстве, но не во всех, компьютерных языков, начинают отсчет первого элемента с индекса 0. Однако иногда хранимые данные имеют больше смысла, если список начинается с 1. (Здесь мы, конечно, не разговариваем с выздоравливающими программистами на Паскале.)

В приведенном ниже решении Джордж Вашингтон является первым президентом, а не нулевым, поэтому если необходимо напечатать список президентов, то проще всего сделать это следующим образом:

```
foreach ($presidents as $number => $president) {
    print "$number: $president\n";
}
```

а не так:

```
foreach ($presidents as $number => $president) {
    $number++;
    print "$number: $president\n";
}
```

Эта функциональность не ограничивается числом 1; годится любое целое:

```
$reconstruction_presidents = array(16 => 'Lincoln', 'Johnson', 'Grant');
```

Кроме того, можно использовать символ => много раз в одном вызове:[1]

```
$whig_presidents = array(9 => 'Harrison', 'Tyler',
                         12 => 'Taylor', 'Fillmore');
```

PHP даже разрешает при вызове функции array() использовать отрицательные числа. (В действительности, этот метод работает также и для нецелых ключей.) То, что дает ассоциативный массив в техническом плане, хотя, как мы сказали, граница между числовым и ассоциативным массивом в PHP размыта, показывает еще один пример, наряду с приведенными ранее.

```
$us_leaders = array(-1 => 'George II', 'George III', 'Washington');
```

Если Вашингтон – это первый президент США, то Джордж III является нулевым, а его дедушка Джордж II – минус первым президентом.

Конечно, можно смешивать и ставить в соответствие числовые и строковые ключи в определении одного массива с помощью функции array(), но это сбивает с толку и используется крайне редко:

```
$presidents = array(1 => 'Washington', 'Adams', 'Honest' =>
                         'Lincoln', 'Jefferson');
```

Это эквивалентно следующему:

```
$presidents[1]        = 'Washington';   // Key is 1
$presidents[]         = 'Adams';        // Key is 1 + 1 => 2
$presidents['Honest'] = 'Lincoln';      // Key is 'Honest'
$presidents[]         = 'Jefferson';    // Key is 2 + 1 => 3
```

См. также

Документацию по функции array() на *http://www.php.net/array*.

4.2. Хранение множества элементов массива с одним ключом

Задача

Необходимо связать различные элементы с одним ключом.

---

[1] Джон Тайлер был избран вице-президентом при президенте Харрисоне от партии вигов, но был исключен из партии сразу после того, как принял на себя обязанности президента после смерти Харрисона.

Решение

Занесение нескольких элементов в массив:

```
$fruits = array('red' => array('strawberry','apple'),
                'yellow' => array('banana'));
```

Или используйте объект:

```
while ($obj = mysql_fetch_object($r)) {
    $fruits[] = $obj;
}
```

Обсуждение

В PHP в пределах одного массива ключи уникальны, поэтому нельзя связать с ключом более одного значения без перезаписи старого значения. Вместо этого запоминают значения в безымянном массиве:

```
$fruits['red'][] = 'strawberry';
$fruits['red'][] = 'apple';
$fruits['yellow'][] = 'banana';
```

Или оперируют с элементами в цикле:

```
while (list($color,$fruit) = mysql_fetch_array($r)) {
    $fruits[$color][] = $fruit;
}
```

Для вывода элементов выполните цикл по всему массиву:

```
foreach ($fruits as $color=>$color_fruit) {
    // $color_fruit - это массив
    foreach ($color_fruit as $fruit) {
        print "$fruit is colored $color.<br>";
    }
}
```

Или используйте функцию `pc_array_to_comma_string()` из рецепта 4.9.

```
foreach ($fruits as $color=>$color_fruit) {
    print "$color colored fruits include " .
        pc_array_to_comma_string($color_fruit) . "<br>";
}
```

См. также

Рецепт 4.9 о том, как распечатывать массивы с запятыми.

4.3. Инициализация массива диапазоном целых чисел

Задача

Необходимо занести в массив ряд последовательных целых чисел.

Решение

Используйте функцию `range($start, $stop)`:

```
$cards = range(1, 52);
```

Обсуждение

Для приращения, отличного от 1, можно использовать:

```
function pc_array_range($start, $stop, $step) {
    $array = array();
    for ($i = $start; $i <= $stop; $i += $step) {
        $array[] = $i;
    }
    return $array;
}
```

Поэтому для нечетных чисел:

```
$odd = pc_array_range(1, 52, 2);
```

А для четных чисел:

```
$even = pc_array_range(2, 52, 2);
```

См. также

Рецепт 2.4 о том, как работать с рядами целых чисел; документацию по функции `range()` на *http://www.php.net/range*.

4.4. Перебор элементов массива

Задача

Необходимо перебрать по очереди и обработать все или некоторые элементы массива.

Решение

Используйте оператор `foreach`:

```
foreach ($array as $value) {
    // Действие с $value
}
```

Или для получения ключей и значений массива:

```
foreach ($array as $key => $value) {
    // Действие II
}
```

Другим способом является применение оператора `for`:

```
for ($key = 0, $size = count($array); $key < $size; $key++) {
```

```
    // Действие III
}
```

И наконец, можно использовать функцию `each()` в комбинации с функцией `list()` и оператором `while`:

```
reset($array) // сброс внутреннего указателя в начало массива
while (list($key, $value) = each ($array)) {
    // Окончательное действие
}
```

Обсуждение

Цикл `foreach` – это наипростейший способ выполнения итераций с массивом:

```
// оператор foreach со значениями
foreach ($items as $cost) {
    ...
}

// оператор foreach с ключами и значениями
foreach($items as $item => $cost) {
    ...
}
```

В операторе `foreach` PHP перебирает не исходный массив, а его копию. Напротив, при использовании функции `each()` и оператора `for`, PHP перебирает оригинальный массив. Поэтому, если внутри цикла происходит модификация массива, то можно получить (а можно и не получить) ожидаемое поведение.

Если необходимо изменить массив, то используйте прямую ссылку на него:

```
reset($items);
while (list($item, $cost) = each($items)) {
    if (! in_stock($item)) {
        unset($items[$item]);           // непосредственная адресация массива
    }
}
```

Переменные, возвращаемые функцией `each()`, не ссылаются на исходные значения массива – это их копии, поэтому их изменение не отражается на массиве. Вот почему нужно модифицировать переменную `$items[$item]` вместо переменной `$item`.

При использовании функции `each()` PHP отслеживает и запоминает положение внутри цикла. Чтобы начать цикл сначала после выполнения первого прохода, нужно вызвать функцию `reset()` для того, чтобы вернуть указатель обратно в положение перед циклом. В противном случае функция `each()` вернет значение `false`.

Цикл `for` работает только в случае массивов с последовательными целыми ключами. Если длина массива не изменяется, то неэффективно

при каждом прохождении цикла снова вызывать функцию `count()` для вычисления переменной `$items`. Поэтому для хранения длины массива всегда используйте переменную `$size`:

```
for ($item = 0, $size = count($items); $item < $size; $item++) {
    ...
}
```

Если вы предпочитаете в целях разумной экономии использовать одну переменную, то считайте в обратном направлении:

```
for ($item = count($items) - 1; $item >= 0; $item--) {
    ...
}
```

Ассоциативная версия цикла `for`:

```
for (reset($array); $key = key($array); next($array) ) {
    ...
}
```

Это приведет к ошибке, если какой-нибудь элемент содержит строку со значением, приравненным к `false`, поэтому вроде бы нормальное значение, такое как 0, может привести к досрочному завершению цикла.

Наконец, используйте функцию `array_map()` для передачи каждого элемента обрабатывающей функции:

```
// переводим все слова в нижний регистр
$lc = array_map('strtolower', $words);
```

Первым аргументом функции `array_map()` является имя функции, которая модифицирует отдельный элемент, а второй аргумент – это массив, обрабатываемый в цикле.

Как правило, эти функции считаются менее гибкими, по сравнению с ранее рассмотренными методами, но они хорошо подходят для обработки и объединения множества массивов.

Если не известно, должны ли обрабатываться данные как скалярные величины или как массив, то необходимо предотвратить использование оператора `foreach` с не массивом. Один из способов – это применение функции `is_array()`:

```
if (is_array($items)) {
    // код с циклом foreach для массива
} else {
    // код для скалярной величины
}
```

Другим способом является принудительное преобразование всех переменных в массив с помощью функции `settype()`:

```
settype($items, 'array');
// код цикла для массивов
```

Это превращает скалярное значение в одноэлементный массив и делает код более привлекательным за счет небольших дополнительных накладных расходов.

См. также

Документацию по оператору `for` на *http://www.php.net/for*, по оператору `foreach` на *http://www.php.net/foreach*, по оператору `while` на *http://www.php.net/while*, по функции `each()` на *http://www.php.net/each*, по функции `reset()` на *http://www.php.net/reset* и по функции `array_map()` на *http://www.php.net/array-map*.

4.5. Удаление элементов из массива

Задача

Необходимо удалить один или более элементов из массива.

Решение

Для удаления одного элемента используйте функцию `unset()`:

```
unset($array[3]);
unset($array['foo']);
```

Для удаления нескольких непоследовательных элементов применяется функция `unset()`:

```
unset($array[3], $array[5]);
unset($array['foo'], $array['bar']);
```

Для удаления нескольких последовательных элементов используйте функцию `array_splice()`:

```
array_splice($array, $offset, $length);
```

Обсуждение

Применение этих функций удаляет все ссылки на эти элементы из PHP. Если необходимо сохранить ключ в массиве, но с пустым значением, присвойте элементу пустую строку:

```
$array[3] = $array['foo'] = '';
```

Помимо синтаксиса есть еще и логическое отличие между использованием функции `unset()` и присваиванием элементу пустой строки (`''`). В первом случае говорится: «Это больше не существует», а во втором – «Это еще существует, но его значение равно пустой строке».

Если мы имеем дело с числами, то присвоение 0 может быть наилучшей альтернативой. Поэтому если компания прекратила производство звездочки модели XL1000, то следующий оператор обновит ее каталог:

```
unset($products['XL1000']);
```

Однако, если компания временно приостановила отпуск звездочки модели XL1000, но планирует получить с завода новую партию позже на этой неделе, это выражение подойдет больше:

```
$products['XL1000'] = 0;
```

После применения функции `unset()` к некоторому элементу PHP корректирует массив так, чтобы цикл продолжал работать правильно. Он не сжимает массив для заполнения пустого пространства. Именно это мы имеем в виду, когда говорим, что все массивы являются ассоциативными, даже если кажутся числовыми. Например:

```
// создаем "нумерованный" массив
$animals = array('ant', 'bee', 'cat', 'dog', 'elk', 'fox');
print $animals[1];  // печатает 'bee'
print $animals[2];  // печатает 'cat'
count($animals);    // возвращает 6

// unset()
unset($animals[1]); // удаляет элемент $animals[1] = 'bee'
print $animals[1];  // печатает '' и выдает ошибку E_NOTICE
print $animals[2];  // все еще печатает 'cat'
count($animals);    // возвращает 5, даже если $array[5] равно 'fox'

// add new element
$animals[] = 'gnu'; // добавляем новый элемент (не в Unix)
print $animals[1];  // печатает '', все еще пустая
print $animals[6];  // печатает 'gnu', где 'gnu' заканчивается
count($animals);    // возвращает 6

// присваиваем ''
$animals[2] = '';   // нулевое выходное значение
print $animals[2];  // печатает ''
count($animals);    // возвращаем 6, счетчик не уменьшается
```

Чтобы сжать массив до плотно заполненного числового массива, используйте функцию `array_values()`:

```
$animals = array_values($animals);
```

В качестве альтернативы функция `array_splice()` автоматически реиндексирует массив, чтобы не оставлять «дыр»:

```
// создаем "числовой" массив
$animals = array('ant', 'bee', 'cat', 'dog', 'elk', 'fox');
array_splice($animals, 2, 2);
print_r($animals);
Array
(
    [0] => ant
    [1] => bee
    [2] => elk
    [3] => fox
)
```

Это полезно, если с массивом работают как с очередью, в то же время разрешая произвольный доступ. Для безопасного удаления первого или последнего элемента массива применяются функции `array_shift()` и `array_pop()` соответственно.

Однако если вы часто сталкиваетесь с проблемами из-за дыр в массивах, возможно, вы не «думаете на PHP». Взгляните на описанные в рецепте 4.4 способы выполнения циклов по массиву, которые не используют цикл `for`.

См. также

Рецепт 4.4 о приемах выполнения итераций; документацию по функции `unset()` на *http://www.php.net/ unset*, по функции `array_splice()` на *http://www.php.net/array-splice* и по функции `array_values()` на *http://www.php.net/array-values*.

4.6. Изменение длины массива

Задача

Необходимо модифицировать длину массива, сделав его больше или меньше текущей длины.

Решение

Для расширения массива применяется функция `array_pad()`:

```
// начинаем с трех
$array = array('apple', 'banana', 'coconut');

// увеличиваем до пяти
$array = array_pad($array, 5, '');
```

Теперь значение функции `count($array)` равно `5`, а последние два элемента содержат пустые строки.

Чтобы сократить массив, можно использовать функцию `array_splice()`:

```
// нет присваивания массиву $array
array_splice($array, 2);
```

Из массива `$array` удаляется все, за исключением двух элементов.

Обсуждение

Размер массивов в PHP заранее не объявляется, поэтому можно менять его по ходу дела.

Для заполнения массива используйте функцию `array_pad()`. В качестве первого аргумента выступает сам заполняемый массив. Следующий аргумент – это размер и направление заполнения. Для заполнения массива вправо используйте положительное число; для заполнения

массива влево используйте отрицательное число. Третий аргумент – это значение, присваиваемое вновь созданным элементам. Функция возвращает модифицированный массив и не затрагивает исходный.

Ниже приведено несколько примеров:

```
// создаем четырехэлементный массив со значением 'dates' справа
$array = array('apple', 'banana', 'coconut');
$array = array_pad($array, 4, 'dates');
print_r($array);
Array
(
    [0] => apple
    [1] => banana
    [2] => coconut
    [3] => dates
)
// создаем шестиэлементный массив со значением 'zucchinis' слева
$array = array_pad($array, -6, 'zucchini');
print_r($array);
Array
(
    [0] => zucchini
    [1] => zucchini
    [2] => apple
    [3] => banana
    [4] => coconut
    [5] => dates
)
```

Будьте внимательны. Выражение `array_pad($array, 4, 'dates')` обеспечивает длину массива, по крайней мере, равную 4, а не добавляет 4 новых элемента. В этом случае, если массив `$array` уже содержал четыре или больше элементов, то функция `array_pad()` возвратит неизмененный массив `$array`.

Также, если объявить значение для четвертого элемента, `$array[4]`:

```
$array = array('apple', 'banana', 'coconut');
$array[4] = 'dates';
```

то в результате получится четырехэлементный массив с индексами `0`, `1`, `2`, and `4`:

```
Array
(
    [0] => apple
    [1] => banana
    [2] => coconut
    [4] => dates
)
```

По существу, PHP превращает его в ассоциативный массив, который, оказывается, имеет целочисленные ключи.

Функция array_splice(), в противоположность функции array_pad(), изменяет исходный массив с одной или с другой стороны. Она возвращает полученный после наложения изменения массив. Вот почему массиву $array не присваивается значение. Однако, как и в случае с функцией array_pad(), можно применить наложение справа или слева. Поэтому вызов функции array_splice() со значением -2 удалит два элемента с конца:

```
// создаем четырехэлементный массив
$array = array('apple', 'banana', 'coconut', 'dates');

// сокращаем до трех элементов
array_splice($array, 3);

// удаляем последний элемент, эквивалентно вызову функции array_pop()
array_splice($array, -1);

// единственные оставшиеся фрукты - это яблоко и банан
print_r($array);
Array
(
    [0] => apple
    [1] => banana
)
```

См. также

Документацию по функции array_pad() на *http://www.php.net/array-pad* и по функции array_splice() на *http: //www.php.net/array-splice*.

4.7. Добавление одного массива к другому

Задача

Необходимо объединить два массива в один.

Решение

Используйте функцию array_merge():

```
$garden = array_merge($fruits, $vegetables);
```

Обсуждение

Функция array_merge() работает и с заранее определенными массивами и с массивами, определенными на месте с помощью функции array():

```
$p_languages = array('Perl', 'PHP');
$p_languages = array_merge($p_languages, array('Python'));
print_r($p_languages);
Array
(
```

```
    [0] => PHP
    [1] => Perl
    [2] => Python
)
```

Соответственно, соединенными массивами могут стать или существующие массивы, как в случае с `$p_languages`, или безымянные массивы, как в случае с `array('Python')`.

Нельзя использовать функцию `array_push()`, поскольку PHP не будет автоматически раскладывать массив в последовательность независимых переменных, и в результате получится вложенный массив. Так:

```
array_push($p_languages, array('Python'));
print_r($p_languages);
Array
(
    [0] => PHP
    [1] => Perl
    [2] => Array
        (
            [0] => Python
        )

)
```

Соединение массивов только с числовыми ключами приводит к перенумерации массивов, поэтому значения не теряются. Объединение массивов со строковыми ключами приведет к тому, что второй массив перепишет значения для всех двойных ключей. Массивы с обоими типами ключей наследуют оба типа поведения. Например:

```
$lc = array('a', 'b' => 'b'); // буквы в нижнем регистре в качестве значений
$uc = array('A', 'b' => 'B'); // буквы в верхнем регистре в качестве значений
$ac = array_merge($lc, $uc);  // все регистры?
print_r($ac);
Array
(
    [0] => a
    [b] => B
    [1] => A
)
```

Буква «A» в верхнем регистре поменяла свой индекс с 0 на 1, чтобы избежать коллизий, и добавилась в конец. Буква «B» в верхнем регистре переписала букву «b» и встала на ее исходное место внутри массива.

С помощью оператора `+` также можно соединять массивы. Массив с правой стороны переписывает любой аналогично названный ключ, найденный в массиве слева. Не делается никакого переупорядочения для предотвращения коллизий. Используем предыдущий пример:

```
print_r($a + $b);
print_r($b + $a);
```

```
Array
(
    [0] => a
    [b] => b
)
Array
(
    [0] => A
    [b] => B
)
```

Так как a и A обе имеют ключ 0, а b и B обе имеют ключ b, то в результате в объединенном массиве будут только два элемента.

В первом случае $a + $b превращается только в $b, а в другом случае $b + $a становится $a.

Однако если бы оба массивы были снабжены очевидными ключами, то проблемы бы не было, и новый массив был бы объединением двух массивов.

См. также

Документацию по функции array_merge() на *http://www.php.net/array-merge*.

4.8. Преобразование массива в строку

Задача

Есть массив, который необходимо конвертировать в хорошо отформатированную строку.

Решение

Используйте функцию join():

```
// создаем список элементов, разделенных запятыми
$string = join(',', $array);
```

Или цикл:

```
$string = '';

foreach ($array as $key => $value) {
    $string .= ",$value";
}

$string = substr($string, 1); // удаляем ведущую ","
```

Обсуждение

Если можно использовать функцию join(), используйте; она работает быстрее, чем любой цикл PHP. Однако функция join() не очень гиб-

кая. Во-первых, она помещает разделитель только между переменными, а не вокруг них. Чтобы вставить элементы внутрь HTML-тегов полужирного начертания текста и разделить их запятыми, сделайте следующее:

```
$left  = '<b>';
$right = '</b>';

$html = $left . join("$right,$left", $html) . $right;
```

Во-вторых, функция `join()` не позволяет различать значения между собой. Если необходимо вставить подмножество значений, нужно использовать собственно цикл:

```
$string = '';

foreach ($fields as $key => $value) {
    // не включаем пароль
    if ('password' != $key) {
        $string .= ",<b>$value</b>";
    }
}

$string = substr($string, 1); // удаляем ведущую ","
```

Обратите внимание, что разделитель всегда добавляется к каждому значению, а затем удаляется вне цикла. И хотя до некоторой степени расточительно сначала добавлять что-нибудь, а потом это же отнимать, но данный прием более привлекательный и эффективный (в большинстве случаев), чем попытка вставить логику внутрь цикла. То есть:

```
$string = '';
foreach ($fields as $key => $value) {
    // не включаем пароль
    if ('password' != $value) {
        if (!empty($string)) { $string .= ','; }
        $string .= "<b>$value</b>";
    }
}
```

Теперь нужно проверять переменную `$string` каждый раз, когда добавляется значение. Это хуже, чем простой вызов функции `substr()`. А также вставлять разделитель (в данном случае запятую) в начало, вместо добавления его, потому что быстрее обрезать строку спереди, чем сзади.

См. также

Рецепт 4.9 о том, как печатать массивы с запятыми; документацию по функции `join()` на *http://www.php.net/join* и по функции `substr()` на *http://www.php.net/substr*.

4.9. Печать массивов с запятыми

Задача

Необходимо распечатать массив с запятыми, разделяющими элементы, и со строкой «and» перед последним элементом, если в массиве больше двух элементов.

Решение

Используйте функцию `pc_array_to_comma_string()`, показанную в примере 4.1, которая возвращает правильную строку.

Пример 4.1. pc_array_to_comma_string( )

```
function pc_array_to_comma_string($array) {

    switch (count($array)) {
    case 0:
        return '';

    case 1:
        return reset($array);

    case 2:
        return join(' and ', $array);

    default:
        $last = array_pop($array);
        return join(', ', $array) . ", and $last";
    }
}
```

Обсуждение

Если необходимо распечатать список элементов, то нелишне печатать их в корректном с точки зрения грамматики стиле. Неуклюже выглядит на экране текст, подобный следующему:

```
$thundercats = array('Lion-O', 'Panthro', 'Tygra', 'Cheetara', 'Snarf');
print 'ThunderCat good guys include ' . join(', ', $thundercats) . '.';
ThunderCat good guys include Lion-O, Panthro, Tygra, Cheetara, Snarf.
```

Такая реализация этой функции не совсем то, что нужно, так как мы хотели, чтобы функция `pc_array_to_comma_string()` работала со всеми массивами, а не только с числовыми, начинающимися с `0`. Если она ограничивается только этим подмножеством, то для одноэлементного массива вернет `$array[0]`. Но если массив начинается не с `0`, то элемент `$array[0]` пустой. Поэтому можно использовать тот факт, что функция `reset()`, которая сбрасывает внутренний указатель массива, также возвращает значение первого элемента массива.

По сходной причине для извлечения последнего элемента вызывается функция `array_pop()` вместо представления его в виде `$array[count($array)-1]`. Это позволяет использовать функцию `join()` для массива `$array`.

Также обратите внимание, что код в `case 2` приведенного выше оператора `switch` в действительности также корректно работает и в `case 1`. А код в `default` работает (хотя и неэффективно) в `case 2`; однако свойство транзитивности не действует, поэтому нельзя применить код по умолчанию к одноэлементным массивам.

См. также

Рецепт 4.8 о том, как превращать массив в строку; документацию по функции `join()` на *http://www.php.net/join*, по функции `array_pop()` на *http://www.php.net/array-pop* и по функции `reset()` на *http://www.php.net/reset*.

4.10. Проверка наличия ключа в массиве

Задача

Необходимо узнать, содержит ли массив определенный ключ.

Решение

Используйте функцию `isset()`:

```
if (isset($array['key'])) { /* В массиве $array есть значение
                               для ключа 'key' */ }
```

Обсуждение

Можно проверить корректность определения элемента массива точно так же, как это делается для любой другой переменной. Более подробную информацию об истинных значениях переменных см. во введении главы 5.

См. также

Документацию по функции `isset()` на *http://www.php.net/isset*.

4.11. Проверка наличия элемента в массиве

Задача

Необходимо узнать, содержит ли массив определенное значение.

Решение

Используйте функцию `in_array()`:

```
if (in_array($array, $value)) {
    // в массиве $array есть элемент со значением $value
}
```

Обсуждение

Используйте функцию `in_array()`, чтобы проверить, содержит ли элемент массива значение:

```
$book_collection = array('Emma', 'Pride and Prejudice', 'Northhanger Abbey');
$book = 'Sense and Sensibility';

if (in_array($book_collection, $book)) {
    echo 'Own it.';
} else {
    echo 'Need it.';
}
```

По умолчанию функция `in_array()` сравнивает данные при помощи оператора `==`. Чтобы провести проверку с оператором строгого равенства `===`, передайте функции `in_array()` значение `true` в качестве третьего параметра:

```
$array = array(1, '2', 'three');

in_array(0, $array);        // true!
in_array(0, $array, true);  // false
in_array(1, $array);        // true
in_array(1, $array, true);  // true
in_array(2, $array);        // true
in_array(2, $array, true);  // false
```

В первой проверке функция `in_array(0, $array)` возвращает `true`, поскольку для сравнения числа `0` со строкой `three` PHP приводит строку `three` к целому значению. А так как строка `three` не является числовой строкой, такой как `2`, она превращается в `0`. Поэтому функция `in_array()` считает, что значения совпадают.

Следовательно, при сравнении чисел с данными, которые могут содержать строки, безопаснее использовать строгое сравнение.

Если функция `in_array()` много раз применяется к тому же самому массиву, возможно лучше использовать ассоциативный массив, в котором ключами являются элементы исходного массива. При использовании функции `in_array()` время поиска меняется по линейному закону, а в случае ассоциативного массива поиск занимает одинаковое время.

Если нельзя создать ассоциативный массив непосредственно, а требуется получить его, конвертируя обычный массив с целочисленными ключами, используйте для замены ключей и значений массива функцию `array_flip()`:

```
$book_collection = array('Emma',
                         'Pride and Prejudice',
                         'Northhanger Abbey');

// преобразование из числового массива в ассоциативный
$book_collection = array_flip($book_collection);
$book = 'Sense and Sensibility';
```

```
if (isset($book_collection[$book])) {
    echo 'Own it.';
} else {
    echo 'Need it.';
}
```

Обратите внимание, что в процессе получения преобразованного массива множество ключей с одинаковым значением сжимаются в один элемент.

См. также

Рецепт 4.12 о том, как определять положение элемента в массиве; документацию по функции `in_array()` на *http://www.php.net/in-array* и по функции `array_flip()` на *http://www.php.net/array-flip*.

4.12. Определение позиции элемента в массиве

Задача

Необходимо узнать, присутствует ли элемент в массиве, и если да, то в какой позиции он находится.

Решение

Используйте функцию `array_search()`. Она возвращает ключ обнаруженного элемента или значение `false`:

```
$position = array_search($array, $value);
if ($position !== false) {
    //элемент массива $array в позиции $position имеет значение $value
}
```

Обсуждение

Используйте функцию `in_array()` для установления наличия в массиве определенного значения; используйте функцию `array_search()` для определения местоположения этого значения. Однако поскольку функция `array_search()` довольно изящно обрабатывает ситуации, когда значение не найдено, то вместо функции `in_array()` лучше использовать функцию `array_search()`. Разница в скорости работы незначительная, а дополнительная информация может оказаться полезной:

```
$favorite_foods = array(1 => 'artichokes', 'bread', 'cauliflower',
                           'deviled eggs');
$food = 'cauliflower';
$position = array_search($food, $favorite_foods);

if ($position !== false) {
    echo "My #$position favorite food is $food";
} else {
```

```
    echo "Blech! I hate $food!";
}
```

Используйте оператор `!==` для сравнения со значением `false`, поскольку если обнаруженная строка находится на нулевой позиции, то оператор `if` преобразует его в логическое значение `false`, что явно не то, что ожидалось.

Если значение встречается в массиве несколько раз, то единственное, что гарантирует функция `array_search()`, — это возвращение одного значения, но не обязательно первого по порядку.

См. также

Рецепт 4.11 о том, как проверять наличие элемента в массиве; документацию по функции `array_search()` на *http://www.php.net/array-search*; для более сложного поиска в массиве с помощью регулярных выражений смотрите информацию по функции `preg_replace()` на *http://www.php.net/preg-replace* и главу 13.

4.13. Нахождение элементов, удовлетворяющих определенному критерию

Задача

Необходимо установить местоположение элементов в массиве, которые удовлетворяют определенным требованиям.

Решение

Используйте цикл `foreach`:

```
$movies = array(...);

foreach ($movies as $movie) {
    if ($movie['box_office_gross'] < 5000000) { $flops[] = $movie; }
}
```

Или функцию `array_filter()`:

```
$movies = array(...);

function flops($movie) {
    return ($movie['box_office_gross'] < 5000000) ? 1 : 0;
}

$flops = array_filter($movies, 'flops');
```

Обсуждение

Цикл `foreach` прост — вы прокручиваете данные и добавляете в возвращаемый массив элементы, которые удовлетворяют вашему критерию.

Если нужен только первый такой элемент, то используйте `break` для выхода из цикла:

```
foreach ($movies as $movie) {
    if ($movie['box_office_gross'] > 200000000) { $blockbuster = $movie;
break; }
}
```

Можно также выйти прямо из функции:

```
function blockbuster($movies) {
    foreach ($movies as $movie) {
        if ($movie['box_office_gross'] > 200000000) { return $movie; }
    }
}
```

Однако при использовании функции `array_filter()` сначала нужно создать функцию обратного вызова, которая возвращает `true` для значений, которые нужно сохранить, и `false` в противном случае. После вызова функции `array_filter()` нужно, чтобы PHP обработал массив таким же образом, как он обрабатывался в операторе цикла `foreach`.

Из функции `array_filter()` невозможно выйти раньше времени, поэтому оператор `foreach` предоставляет большую гибкость и его проще понять. К тому же это один из немногих случаев, когда встроенная функция PHP не имеет явного превосходства над кодом пользовательского уровня.

См. также

Документацию по функции `array_filter()` на *http://www.php.net/array-filter*.

4.14. Нахождение элемента массива с наибольшим или наименьшим значением

Задача

Есть массив элементов, и необходимо найти элемент с наибольшим или наименьшим значением. Например, нужно определить соответствующий масштаб при создании гистограммы.

Решение

Для нахождения наибольшего элемента используйте функцию `max()`:

```
$largest = max($array);
```

Для нахождения наименьшего элемента используйте функцию `min()`:

```
$smallest = min($array);
```

Обсуждение

Обычно функция `max()` возвращает наибольший из двух элементов, но если ей передается массив, то она осуществляет поиск среди элементов массива. К сожалению, при использовании функции `max()` нельзя узнать индекс наибольшего элемента. Чтобы это сделать, необходимо отсортировать массив в порядке убывания, поместив наибольший элемент в нулевую позицию:

```
arsort($array);
```

Теперь значение наибольшего элемента находится в `$array[0]`.

Если не хотите затрагивать порядок исходного массива, то сделайте копию и отсортируйте ее:

```
$copy = $array;
arsort($copy);
```

Та же идея применима и к функции `min()`, но вместо функции `arsort()` используйте функцию `asort()`.

См. также

Рецепт 4.16 о сортировке массива; документацию по функции `max()` на *http://www.php.net/max*, по функции `min()` на *http://www.php.net/min*, по функции `arsort()` на *http://www.php.net/arsort* и по функции `asort()` на *http://www.php.net/asort*.

4.15. Обращение массива

Задача

Необходимо изменить порядок расположения элементов массива на обратный.

Решение

Используйте функцию `array_reverse()`:

```
$array = array('Zero', 'One', 'Two');
$reversed = array_reverse($array);
```

Обсуждение

Функция `array_reverse()` изменяет порядок следования элементов массива на обратный. Однако часто можно избежать этой операции. Если нужно обратить массив, просто отсортируйте его, изменив порядок сортировки на обратный. Если нужно перевернуть список, который обрабатывается в цикле, просто инвертируйте цикл. Вместо:

```
for ($i = 0, $size = count($array); $i < $size; $i++) {
    ...
}
```

делайте так:

```
for ($i = count($array) - 1; $i >=0 ; $i--) {
    ...
}
```

Однако, как всегда, применяйте цикл `for` только для плотно упакованных массивов.

Другой альтернативой изменения порядка элементов может быть размещение их в массиве. Например, при заполнении массива рядом строк из базы данных можно модифицировать запрос с помощью выражения `ORDER DESC`. Уточнить синтаксис можно в руководстве по базе данных.

См. также

Документацию по функции `array_reverse()` на *http://www.php.net/array-reverse*.

4.16. Сортировка массива

Задача

Необходимо отсортировать массив определенным образом.

Решение

Для сортировки массива в общепринятом смысле этого слова используйте функцию `sort()`:

```
$states = array('Delaware', 'Pennsylvania', 'New Jersey');
sort($states);
```

Для сортировки в числовом порядке передайте `SORT_NUMERIC` в качестве второго аргумента функции `sort()`.

```
$scores = array(1, 10, 2, 20);
sort($scores, SORT_NUMERIC);
```

Числа будут отсортированы в порядке возрастания `(1, 2, 10, 20)` вместо лексикографического порядка `(1, 10, 2, 20)`.

Обсуждение

Функция `sort()` не сохраняет связи ключ/значение между элементами; вместо этого она реиндексирует элементы, начиная с `0` по возрастанию. (Единственным исключением является одноэлементный массив; индекс его единственного элемента не сбрасывается в `0`. Это исправлено, начиная с версии PHP 4.2.3.)

Для сохранения связей ключ/значение используйте функцию `asort()`. Функция `asort()` обычно используется для ассоциативных массивов,

но она может оказаться полезной и в случае, когда индексы элементов имеют смысловое содержание:

```
$states = array(1 => 'Delaware', 'Pennsylvania', 'New Jersey');
asort($states);

while (list($rank, $state) = each($states)) {
    print "$state was the #$rank state to join the United States\n";

}
```

Используйте функцию `natsort()` для упорядочения массива по естественному алгоритму сортировки. При естественной сортировке можно смешивать строки и числа внутри элементов и получать при этом правильный результат.

```
$tests = array('test1.php', 'test10.php', 'test11.php', 'test2.php');
natsort($tests);
```

Теперь элементы расположены по порядку: `'test1.php'`, `'test2.php'`, `'test10.php'` и `'test11.php'`. При естественной сортировке число 10 располагается после числа 2; обычная сортировка приведет к противоположному результату. Для выполнения нечувствительной к регистру естественной сортировки используйте функцию `natcasesort()`.

Чтобы отсортировать массив в обратном порядке, используйте функцию `rsort()` или функцию `arsort()`, которая действует подобно функции `rsort()`, но к тому же сохраняет ключи. Не существует функции `natrsort()` или `natcasersort()`. В эти функции можно также передать в качестве аргумента выражение `SORT_NUMERIC`.

См. также

Рецепт 4.17 о сортировке с помощью пользовательской функции и рецепт 4.18 о сортировке множества массивов; документацию по функции `sort()` на *http://www.php.net/sort*, по функции `asort()` на *http://www.php.net/asort*, по функции `natsort()` на *http://www.php.net/natsort*, по функции `natcasesort()` на *http://www.php.net/natcasesort*, по функции `rsort()` на *http://www.php.net/rsort* и по функции `arsort()` на *http://www.php.net/arsort*.

4.17. Сортировка массива по вычисляемому полю

Задача

Необходимо задать собственную процедуру сортировки.

Решение

Используйте функцию `usort()` в комбинации с пользовательской функцией:

```
// сортируем в порядке, обратном естественному
function natrsort($a, $b) {
    return strnatcmp($b, $a);
}

$tests = array('test1.php', 'test10.php', 'test11.php', 'test2.php');
usort($tests, 'natrsort');
```

Обсуждение

Функция сравнения должна возвращать значение больше `0`, если `$a > $b`, равное `0`, если `$a == $b` и значение меньше `0`, если `$a < $b`. Для сортировки в обратном порядке делайте наоборот. Функция `strnatcmp()`, приведенная в разделе «Решение», подчиняется этим правилам.

Чтобы поменять направление сортировки на обратное, вместо умножения возвращаемого значения `strnatcmp($a, $b)` на `-1` поменяйте местами аргументы в функции `strnatcmp($b, $a)`.

Функции сортировки не обязательно должны быть оболочками существующей сортировки. Например, функция `pc_date_sort()`, приведенная в примере 4.2, показывает, как сортировать даты.

Пример 4.2. pc_date_sort( )

```
// ожидаем дату в формате "MM/DD/YYYY"
function pc_date_sort($a, $b) {
    list($a_month, $a_day, $a_year) = explode('/', $a);
    list($b_month, $b_day, $b_year) = explode('/', $b);

    if ($a_year  > $b_year ) return  1;
    if ($a_year  < $b_year ) return -1;

    if ($a_month > $b_month) return  1;
    if ($a_month < $b_month) return -1;

    if ($a_day   > $b_day  ) return  1;
    if ($a_day   < $b_day  ) return -1;

    return 0;
}

$dates = array('12/14/2000', '08/10/2001', '08/07/1999');
usort($dates, 'pc_date_sort');
```

Во время сортировки функция `usort()` часто – каждый раз, когда ей надо сравнить два элемента – выполняет пересчет значений, возвращаемых функцией сортировки, что замедляет процесс. Для того чтобы избежать ненужной работы, можно кэшировать сравниваемые значения, как показано в примере 4.3.

Пример 4.3. pc_array_sort( )

```
function pc_array_sort($array, $map_func, $sort_func = '') {
    $mapped = array_map($map_func, $array);  // cache $map_func() values

    if ('' == $sort_func) {
```

```
        asort($mapped);        // функция asort() быстрее функции usort()
    }  else {
        uasort($mapped, $sort_func); // необходимо сохранить ключи
    }

    while (list($key) = each($mapped)) {
        $sorted[] = $array[$key];    // используем отсортированные ключи
    }

    return $sorted;
}
```

Чтобы избежать ненужной работы, функция `pc_array_sort()` использует временный массив `$mapped` для кэширования возвращаемых значений. Затем она сортирует массив `$mapped`, используя или порядок сортировки по умолчанию, или определенную пользователем процедуру сортировки. Важно, что она использует сортировку, сохраняющую связи ключ/значение. По умолчанию она использует функцию `asort()`, потому что она быстрее, чем функция `uasort()`. (Медленность функции `uasort()` это все-таки значительный довод в пользу функции `pc_array_sort()`.) Наконец, она создает отсортированный массив `$sorted`, при этом отсортированные ключи в массиве `$mapped` выступают в качестве индексов значений исходного массива.

Для небольших массивов или коротких функций сортировки функция `usort()` работает быстрее, но как только число сравнений вырастает, функция `pc_array_sort()` обгоняет функцию `usort()`. В следующем примере элементы сортируются по длине их строки (это относительно быстрая пользовательская сортировка):

```
function pc_u_length($a, $b) {
    $a = strlen($a);
    $b = strlen($b);

    if ($a == $b) return  0;
    if ($a  > $b) return  1;
                  return -1;
}

function pc_map_length($a) {
    return strlen($a);
}

$tests = array('one', 'two', 'three', 'four', 'five',
               'six', 'seven', 'eight', 'nine', 'ten');

// для количества элементов < 5 функция pc_u_length() быстрее
usort($tests, 'pc_u_length');

// для количества элементов >= 5 функция pc_map_length() быстрее
$tests = pc_array_sort($tests, 'pc_map_length');
```

Здесь функция `pc_array_sort()` быстрее функции `usort()`, поскольку массив достигает длины в пять элементов.

См. также

Рецепт 4.16 об основах сортировки и рецепт 4.18 о сортировке множества массивов; документацию по функции `usort()` на *http://www.php.net/usort*, по функции `asort()` на *http://www.php.net/asort* и по функции `array_map()` на *http://www.php.net/array-map*.

4.18. Сортировка множества массивов

Задача

Необходимо отсортировать несколько массивов или многомерный массив.

Решение

Используйте функцию `array_multisort()`:

Чтобы одновременно отсортировать несколько массивов, передайте это множество массивов функции `array_multisort()`:

```
$colors = array('Red', 'White', 'Blue');
$cities = array('Boston', 'New York', 'Chicago');

array_multisort($colors, $cities);
print_r($colors);
print_r($cities);
Array
(
    [0] => Blue
    [1] => Red
    [2] => White
)
Array
(
    [0] => Chicago
    [1] => Boston
    [2] => New York
)
```

Чтобы отсортировать несколько измерений внутри одного массива, передайте определенные элементы массива:

```
$stuff = array('colors' => array('Red', 'White', 'Blue'),
               'cities' => array('Boston', 'New York', 'Chicago'));

array_multisort($stuff['colors'], $stuff['cities']);
print_r($stuff);
Array
(
    [colors] => Array
        (
            [0] => Blue
            [1] => Red
```

```
                [2] => White
            )
        [cities] => Array
            (
                [0] => Chicago
                [1] => Boston
                [2] => New York
            )
    )
```

Чтобы изменить тип сортировки, как в функции `sort()`, передайте `SORT_REGULAR`, `SORT_NUMERIC` или `SORT_STRING` после имени массива. Для изменения порядка сортировки, в отличие от функции `sort()`, передайте `SORT_ASC` или `SORT_DESC` после имени массива. Можно передать как тип, так и порядок сортировки.

Обсуждение

Функция `array_multisort()` может сортировать несколько массивов одновременно или многомерный массив по одному или более направлений. Массивы трактуются как колонки таблицы, которая должна быть отсортирована по строкам. Первый массив является главным массивом для сортировки; порядок всех элементов других массивов устанавливается на основе порядка сортировки первого массива. Если элементы первого массива равны, то их порядок определяется вторым массивом, и т. д.

По умолчанию устанавливаются значения сортировки `SORT_REGULAR` и `SORT_ASC`, и они переустанавливаются после каждого массива, поэтому нет необходимости передавать какое-либо из этих значений, разве что для ясности.

```
$numbers = array(0, 1, 2, 3);
$letters = array('a', 'b', 'c', 'd');
array_multisort($numbers, SORT_NUMERIC, SORT_DESC,
                $letters, SORT_STRING , SORT_DESC);
```

Код этого примера обращает массивы.

См. также

Рецепт 4.16 о простой сортировке и рецепт 4.17 о сортировке с помощью пользовательской функции; документацию по функции `array_multisort()` на *http://www.php.net/array-multisort*.

4.19. Сортировка массива с использованием метода вместо функции

Задача

Необходимо определить пользовательскую процедуру сортировки массива. Однако вместо функции нужно применить метод объекта.

Решение

Передайте массив, содержащий имя класса и метода вместо имени функции:

```
usort($access_times, array('dates', 'compare'));
```

Обсуждение

Как и в случае с пользовательской функцией, метод объекта должен принять два входных аргумента, а возвратить значение 1, 0 или -1 – в зависимости от того, больше ли первый аргумент второго, равен ли ему или меньше:

```
class pc_sort {
    // обратный порядок сравнения строки
    function strrcmp($a, $b) {
        return strcmp($b, $a);
    }
}

usort($words, array('pc_sort', 'strrcmp'));
```

См. также

Дополнительную информацию о классах и объектах в главе 7; дополнительные сведения о пользовательской сортировке массивов в рецепте 4.17.

4.20. Рандомизация массива

Задача

Необходимо перетасовать элементы массива в случайном порядке.

Решение

Если у вас запущена версия PHP 4.3 или выше, то используйте функцию `shuffle()`:

```
shuffle($array);
```

С более ранними версиями используйте функцию `pc_array_shuffle()`, показанную в примере 4.4.

Пример 4.4. pc_array_shuffle( )

```
function pc_array_shuffle($array) {
    $i = count($array);

    while(--$i) {
        $j = mt_rand(0, $i);

        if ($i != $j) {
```

```
            // перестановка элементов
            $tmp = $array[$j];
            $array[$j] = $array[$i];
            $array[$i] = $tmp;
        }
    }

    return $array;
}
```

Ниже приведены примеры:

```
$cards = range(1,52); // deal out 52 "cards"
$cards = pc_array_shuffle($cards);
```

Обсуждение

В PHP уже существует функция `shuffle()` для перемешивания массивов, однако в PHP 4.2.2 она работает некорректно. Встроенный алгоритм перемешивания имеет тенденцию давать предпочтение одним определенным перестановкам перед другими. Последние перестановки выглядят случайными, но так как элементы в каждой конкретной позиции имеют разные шансы оказаться в конце процесса, то такая перетасовка является недостоверной. Это исправлено в PHP 4.3.

Функция `pc_array_shuffle()`, известная как перестановка Фишера-Йетса, одинаково распределяет элементы вдоль массива. Используйте ее с версиями PHP более ранними, чем 4.3. В отличие от `shuffle( )`, эта функция возвращает перемешанный массив, а не изменяет его по месту. Она также требует плотно упакованного массива с целочисленными ключами.

См. также

Рецепт 4.21 о функции, которая моделирует тасование колоды карт; документацию по функции `shuffle()` на *http://www.php.net/shuffle*.

4.21. Тасование колоды карт

Задача

Необходимо перетасовать колоду карт и раздать их.

Решение

Создайте массив из 52 целых чисел, перемешайте его и свяжите с картами:

```
$suits = array('Clubs', 'Diamonds', 'Hearts', 'Spades');
$cards = array('Ace', 2, 3, 4, 5, 6, 7, 8, 9, 10, 'Jack', 'Queen', 'King');

$deck = pc_array_shuffle(range(0, 51));
```

```
while (($draw = array_pop($deck)) != NULL) {
    print  $cards[$draw / 4] . ' of ' . $suits[$draw % 4] . "\n";
}
```

Этот код использует функцию `pc_array_shuffle()` из рецепта 4.20.

Обсуждение

Для английского представления карт создается пара массивов, `$suits` и `$cards`. Числа от `0` до `51` случайным образом расставляются и назначаются массиву `$deck`. Чтобы сдать карту, достаточно извлечь число из начала массива, рассматривая этот массив как буквальную колоду карт.

Необходимо добавить проверку на значение `NULL` внутри оператора `while`, иначе цикл прервется, когда вы вытащите нулевую карту. Если изменить колоду так, чтобы она содержала числа от 1 до 52, то, с математической точки зрения, сопоставление чисел и карт становится более сложным.

Чтобы сдать несколько карт сразу, вызовите функцию `array_slice()`:

```
array_slice($deck, $cards * -1);
```

См. также

Рецепт 4.20 о функции, которая рандомизирует массив; документацию по функции `array_slice()` на *http://www.php.net/array-slice*.

4.22. Удаление двойных элементов из массива

Задача

Необходимо удалить дубликаты из массива.

Решение

Если массив уже заполнен, используйте функцию `array_unique()`, которая возвращает новый массив, не содержащий двойных значений:

```
$unique = array_unique($array);
```

Ниже показан способ получения необходимого результата в процессе создания числовых массивов:

```
foreach ($_REQUEST['fruits'] as $fruit) {
    if (!in_array($array, $fruit)) { $array[] = $fruit; }
}
```

Тот же метод для ассоциативных массивов:

```
foreach ($_REQUEST['fruits'] as $fruit) {
    $array[$fruit] = $fruit;
}
```

Обсуждение

Если процесс завершен, то применение функции `array_unique()` является лучшим способом удаления дубликатов. Но если вы внутри цикла, то предупредить появление двойных элементов можно с помощью проверки, не находятся ли уже эти элементы в массиве.

Создание гибридного массива, в котором ключ и значение каждого элемента одинаковы, является методом даже более быстрым, чем использование функции `in_array()`. Это исключает линейную проверку с помощью функции `in_array()`, но в то же время позволяет воспользоваться преимуществами семейства функций для работы с массивами, которые оперируют со значениями массива, а не с ключами.

В действительности, быстрее использовать метод ассоциативного массива, а затем применить функцию `array_values()` к результату (или, коли на то пошло, `array_keys()`, но `array_values()` немного быстрее), чем непосредственно создавать числовой массив через голову функции `in_array()`.

См. также

Документацию по функции `array_unique()` на *http://www.php.net/array-unique*.

4.23. Определение объединения, пересечения или разности двух массивов

Задача

Есть два массива, и требуется найти их объединение (все элементы, но если элемент входит в оба массива, он учитывается один раз), пересечение (элементы, входящие в оба массива) или разность (элементы одного массива, не присутствующие в другом).

Решение

Для определения объединения:

```
$union = array_unique(array_merge($a, $b));
```

Для вычисления пересечения:

```
$intersection = array_intersection($a, $b);
```

Для нахождения простой разности:

```
$difference = array_diff($a, $b);
```

И для получения симметрической разности (исключающее ИЛИ):

```
$difference = array_merge(array_diff($a, $b), array_diff($b, $a));
```

Обсуждение

Многие из необходимых для таких вычислений компонентов встроены в PHP, нужно только объединить их в соответствующей последовательности.

При получении объединения из двух массивов создается один гигантский массив со всеми значениями исходных массивов. Но функция `array_merge()` разрешает дубликаты значений при объединении двух числовых массивов, поэтому нужно вызвать функцию `array_unique()`, чтобы отфильтровать такие элементы. Но при этом могут образоваться пропуски, поскольку функция `array_unique()` не уплотняет массив. Однако это не представляет затруднения, поскольку и оператор `foreach`, и функция `each()` без помех обрабатывают редко заполненные массивы.

Функция для вычисления пересечения имеет простое имя `array_intersection()` и не требует дополнительных усилий.

Функция `array_diff()` возвращает массив, содержащий все уникальные элементы массива `$old`, которые не входят в массив `$new`. Это называется *простой разностью*:

```
$old = array('To', 'be', 'or', 'not', 'to', 'be');
$new = array('To', 'be', 'or', 'whatever');
$difference = array_diff($old, $new);
Array
(
    [3] => not
    [4] => to
)
```

Результирующий массив `$difference` содержит `'not'` и `'to'`, так как функция `array_diff()` чувствительна к регистру. В него не входит элемент `'whatever'`, поскольку его нет в массиве `$old`.

Чтобы получить обратную разность, или, другими словами, найти уникальные элементы массива `$new`, отсутствующие в массиве `$old`, нужно поменять местами аргументы:

```
$old = array('To', 'be', 'or', 'not', 'to', 'be');
$new = array('To', 'be', 'or', 'whatever');
$reverse_diff = array_diff($new, $old);
Array
(
    [3] => whatever
)
```

Массив `$reverse_diff` содержит только элемент `'whatever'`.

Если нужно применить функцию или другой фильтр в функции `array_diff()`, встройте свой собственный алгоритм нахождения разности (вычитания):

```
// применим нечувствительный к регистру алгоритм вычитания; разность -i
```

```
$seen = array();
foreach ($new as $n) {
    $seen[strtolower($n)]++;
}

foreach ($old as $o) {
    $o = strtolower($o);
    if (!$seen[$o]) { $diff[$o] = $o; }
}
```

Первый оператор `foreach` создает ассоциативный массив для дальнейшего поиска. Затем выполняется цикл по массиву `$old` и, если в процессе поиска элемент не найден, то он добавляется в массив `$diff`.

Этот процесс можно ускорить, объединив функции `array_diff()` и `array_map()`:

```
$diff = array_diff(array_map('strtolower', $old),
                   array_map('strtolower', $new));
```

Симметрическая разность – это то, что принадлежит `$a`, но не принадлежит `$b`, плюс то, что есть в `$b`, но нет в `$a`:

```
$difference = array_merge(array_diff($a, $b), array_diff($b, $a));
```

Однажды установленный, алгоритм движется вперед. Функция `array_diff()` вызывается дважды и определяет две разности. Затем они объединяются в один массив. Нет необходимости вызывать функцию `array_unique()`, так как эти массивы были специально сконструированы как не имеющие общих элементов.

См. также

Документацию по функции `array_unique()` на *http://www.php.net/array-unique*, по функции `array_intersect()` на *http://www.php.net/array-intersect*, по функции `array_diff()` на *http://www.php.net/array-diff*, по функции `array_merge()` на *http://www.php.net/array-merge* и по функции `array_map()` на *http://www.php.net/ array-map*.

4.24. Определение всех комбинаций элементов массива

Задача

Необходимо определить совокупность всех комбинаций множеств, содержащих все или некоторые элементы массива, известную также как *показательное множество*.

Решение

Используйте функцию `pc_array_power_set()`, приведенную в примере 4.5.

Пример 4.5. pc_array_power_set( )

```
function pc_array_power_set($array) {
    // инициализируем пустым множеством
    $results = array(array());

    foreach ($array as $element)
        foreach ($results as $combination)
            array_push($results, array_merge(array($element), $combination));

    return $results;
}
```

Она возвращает массив массивов, содержащий все комбинации элементов, включая пустое множество. Например:

```
$set = array('A', 'B', 'C');
$power_set = pc_array_power_set($set);
```

Массив `$power_set` состоит из восьми массивов:

```
array();
array('A');
array('B');
array('C');
array('A', 'B');
array('A', 'C');
array('B', 'C');
array('A', 'B', 'C');
```

Обсуждение

Сначала включаем в результат пустое множество `{}`. Все-таки должна быть одна комбинация множеств, которая не содержит ни одного элемента из них.

Остальная часть этой функции основана на природе комбинаций и реализации оператора `foreach` в PHP. Каждый новый элемент, добавленный в массив, увеличивает число комбинаций. Новые комбинации представляют собой старые комбинации с новым элементом; двухэлементный массив, состоящий из `A` и `B`, генерирует четыре возможных комбинации: `{}`, `{A}`, `{B}` и `{A, B}`. Добавление `C` к этому множеству сохраняет четыре предыдущих комбинации, а также добавляет четыре новых комбинации: `{C}`, `{A, C}`, `{B, C}` и `{A, B, C}`.

Таким образом, внешний цикл `foreach` проходит по всем элементам списка; внутренний цикл `foreach` проходит по всем предыдущим комбинациям, созданным более ранними элементами. Это немного сложно, но необходимо точно знать, как ведет себя PHP в процессе выполнения цикла `foreach`.

Функция `array_merge()` объединяет элемент с предыдущими комбинациями. Заметим, однако, что массив `$results`, добавленный к новому массиву функцией `array_push()`, – это тот самый массив, который обра-

батывался в цикле foreach. Обычно добавление элемента к массиву $results приводит к бесконечному циклу, но не в PHP, поскольку PHP работает с копией исходного списка. И когда вы возвращаетесь на уровень внешнего цикла и снова выполняете цикл foreach со следующим элементом, эта копия сбрасывается. Поэтому можно работать непосредственно с массивом $results и использовать его в качестве стека для хранения комбинаций. Хранение всей информации в массиве даст большую гибкость, когда в дальнейшем придется выводить на печать и дополнительно подразделять на комбинации.

Чтобы исключить пустое множество, замените начальную строку:

```
// инициализируем, добавляя пустое множество
$results = array(array());
```

на:

```
// инициализируем, добавляя первый элемент
$results = array(array(array_pop($array)));
```

Поскольку одноэлементный массив имеет только одну комбинацию – самого себя, то удаление элемента равносильно выполнению первого прохода цикла. Двойные циклы foreach не знают, что в действительности они начинают свою работу со второго элемента массива.

Чтобы напечатать результат с табуляциями между элементами внутри комбинации и возвратом каретки в конце каждой комбинации, используйте следующий код:

```
$array = array('Adam', 'Bret', 'Ceff', 'Dave');

foreach (pc_array_power_set($array) as $combination) {
    print join("\t", $combination) . "\n";
}
```

Ниже показано, как напечатать только трехэлементные комбинации:

```
foreach (pc_array_power_set($set) as $combination) {
    if (3 == count($combination)) {
        print join("\t", $combination) . "\n";
    }
}
```

Итерация с большими множествами занимает значительное время. Множество из n элементов приводит к образованию 2^{n+1} множеств. Другими словами, увеличение n на 1 вызывает удвоение количества элементов.

См. также

Рецепт 4.25 о функции, которая находит все перестановки массива.

4.25. Нахождение всех перестановок массива

Задача

Есть массив элементов, и необходимо вычислить все возможные варианты упорядочения массива.

Решение

Используйте один из двух алгоритмов перестановки, обсуждаемых далее.

Обсуждение

Функция `pc_permute()`, приведенная в примере 4.6, – это PHP-модификация базовой рекурсивной функции.

Пример 4.6. pc_permute( )

```
function pc_permute($items, $perms = array()) {
    if (empty($items)) {
        print join(' ', $perms) . "\n";
    }  else {
        for ($i = count($items) - 1; $i >= 0; --$i) {
             $newitems = $items;
             $newperms = $perms;
             list($foo) = array_splice($newitems, $i, 1);
             array_unshift($newperms, $foo);
             pc_permute($newitems, $newperms);
         }
    }
}
```

Например:

```
pc_permute(split(' ', 'she sells seashells'));
she sells seashells
she seashells sells
sells she seashells
sells seashells she
seashells she sells
seashells sells she
```

Эта рекурсия хотя и элегантна, но неэффективна, поскольку делает повсюду копии. К тому же не так легко модифицировать эту функцию, чтобы она возвращала значения вместо вывода на печать без накопления их в глобальной переменной.

Впрочем, функция `pc_next_permutation()`, показанная в примере 4.7, немного приятнее. Она объединяет идею Марка-Джейсона Доминуса (Mark-Jason Dominus) из «Perl Cookbook» Тома Кристиансена (Tom Christianson) и Натана Торкингтона (Nathan Torkington) (издательство O'Reilly) с алгоритмом из классического труда Эдсгера Дейкстры

(Edsger Dijkstra) «A Discipline of Programming» (издательство Prentice-Hall).

Пример 4.7. pc_next_permutation( )

```
function pc_next_permutation($p, $size) {
    // проходим массив сверху вниз в поисках числа, которое меньше следующего
    for ($i = $size - 1; $p[$i] >= $p[$i+1]; --$i) { }

    // если такого нет, прекращаем перестановки
    // массив перевернут: (1, 2, 3, 4) => (4, 3, 2, 1)
    if ($i == -1) { return false; }

    // проходим массив сверху вниз в поисках числа,
    // превосходящего найденное ранее
    for ($j = $size; $p[$j] <= $p[$i]; --$j) { }

    // переставляем их
    $tmp = $p[$i]; $p[$i] = $p[$j]; $p[$j] = $tmp;

    // теперь переворачиваем массив путем перестановки элементов,
    // начиная с конца
    for (++$i, $j = $size; $i < $j; ++$i, --$j) {
         $tmp = $p[$i]; $p[$i] = $p[$j]; $p[$j] = $tmp;
    }

    return $p;
}

$set = split(' ', 'she sells seashells');   // подобно массиву ('she',
                                               'sells', 'seashells')
$size = count($set) - 1;
$perm = range(0, $size);
$j = 0;

do {
     foreach ($perm as $i) { $perms[$j][] = $set[$i]; }
} while ($perm = pc_next_permutation($perm, $size) and ++$j);

foreach ($perms as $p) {
    print join(' ', $p) . "\n";
}
```

Идея Доминуса состоит в том, что вместо манипуляций с самим массивом можно создавать перестановки целых чисел. Затем, чтобы получить истинную перестановку, снова ставим в соответствие числам элементы массива – оригинальная мысль.

Однако этот прием имеет некоторые недостатки. Для нас, программистов на PHP, более важными являются частые извлечения, вставки и объединения массивов, т. е. то, что для Perl является центральным. Затем процесс вычисления перестановок целых чисел проходит через серию шагов, которые выполняются для каждой перестановки; и поскольку он не запоминает предыдущие перестановки, то каждый раз начинает с исходной перестановки. Зачем работает повторное выполнение, если мы можем помочь ему?

Алгоритм Дейкстры решает это, принимая перестановку ряда целых чисел и возвращая следующую наибольшую перестановку. Код оптимизируется на основе этого предположения. Начиная с наименьшего шаблона (который представлен просто целыми числами, расположенными по возрастанию) и продолжая выполнение снизу вверх, можно прокрутить все перестановки за один раз, передавая предыдущую перестановку обратно в функцию для получения следующей. Едва ли там имеют место хоть какие-то перестановки, даже в последнем цикле перестановки, где переставляется конец.

Метод предоставляет дополнительные преимущества. Рецепт Доминуса требует общего количества перестановок для данного шаблона. Так как оно равно факториалу количества элементов в множестве, то требует довольно больших вычислительных затрат, даже с промежуточным хранением результата. Вместо определения этого числа быстрее вернуть значение `false` из функции `pc_next_permutation()`, если окажется, что `$i == -1`. Когда это происходит, вы вынуждены покинуть массив, исчерпав перестановки данной фразы.

Два последних замечания по реализации. Поскольку размер множества – это константа, то он определяется однажды с помощью функции `count()` и передается в функцию `pc_next_permutation()`; это быстрее, чем повторно вызывать функцию `count()` внутри функции. Кроме того, так как уникальность элементов множества гарантируется самой его структурой, т. е. в нем одно и только одно вхождение каждого целого числа, то нет необходимости проводить проверку на равенство внутри первых двух циклов `for`. Однако это нужно делать при использовании этого рецепта для других числовых множеств, где могут встречаться дубликаты.

См. также

Рецепт 4.24 о функции, которая определяет показательное множество массива; рецепт 4.19 в книге «Perl Cookbook» издательства O'Reilly;[1] глава 3 из книги Дейкстры «A Discipline of Programming» издательства Prentice-Hall.[2]

4.26. Программа: Печать массива в виде HTML-таблицы

Преобразование массива в таблицу с горизонтально расположенными столбцами располагает фиксированное количество элементов в строке.

---

[1] Кристиансен Т., Торкингтон Н. «Perl. Сборник рецептов. Для профессионалов», 2-е издание. – Пер. с англ. – СПб.: Питер, 2004.

[2] Дейкстра Э. «Дисциплина программирования». – Пер. с англ. – М.: Мир, 1978.

Первое множество заполняет начальную строку таблицы, второе множество располагается в следующей строке и так далее. Наконец доходим до последней строки, которую, возможно, придется заполнить пустыми ячейками таблицы.

Функция `pc_grid_horizontal()`, показанная в примере 4.8, позволяет указать массив и число столбцов. Она предполагает ширину таблицы, равную 100%, но ее можно изменить с помощью переменной `$table_width`.

Пример 4.8. pc_grid_horizontal( )

```
function pc_grid_horizontal($array, $size) {

    // вычисляем ширину элемента <td> в процентах
    $table_width = 100;
    $width = intval($table_width / $size);

    // определяем вид тегов <tr> и <td>
    // функция sprintf() требует использования %% для получения символа %
    $tr = '<tr align="center">';
    $td = "<td width=\"$width%%\">%s</td>";

    // открываем таблицу
    $grid = "<table width=\"$table_width%\">$tr";

    // выполняем цикл по элементам и отображаем в строке длиной $sized
    // $i отслеживает, когда нужно начинать новую строку таблицы
    $i = 0;
    foreach ($array as $e) {
        $grid .= sprintf($td, $e);
        $i++;

        // конец строки
        // закрываем ее и начинаем новую
        if (!($i % $size)) {
            $grid .= "</tr>$tr";
        }
    }

    // заполняем остальные ячейки пробелами
    while ($i % $size) {
        $grid .= sprintf($td, ' ');
        $i++;
    }

    // добавляем </tr> при необходимости
    $end_tr_len = strlen($tr) * -1;
    if (substr($grid, $end_tr_len) != $tr) {
        $grid .= '</tr>';
    } else {
        $grid = substr($grid, 0, $end_tr_len);
    }
```

```
    // закрываем таблицу
    $grid .= '</table>';

    return $grid;
}
```

Функция начинается с вычисления ширины каждого элемента <td> в процентах к общей ширине таблицы. В зависимости от количества столбцов и общего размера, сумма ширины элементов <td> может не совпадать с шириной элемента <table>, но это не должно заметно влиять на отображение HTML. Затем определяются теги <td> и <tr>, при этом используется нотация форматирования в стиле функции printf. Для получения символа %, необходимого для выражения в процентах ширины элемента <td>, используйте сдвоенный символ %%.

Ядро функции – это цикл foreach по элементам массива, в котором каждый тег td> добавляется к переменной $grid. При достижении конца строки, что происходит, когда общее число обработанных элементов становится кратным количеству элементов в строке, элемент <tr> закрывается и открывается снова.

После того как добавлены все элементы, необходимо заполнить последнюю строку пробелами или пустыми элементами <td>. Для корректной передачи таблицы в броузер поместите непрерывную пробельную строку в ячейку данных, вместо того чтобы оставлять ее пустой. Теперь убедитесь в отсутствии лишних элементов <tr> в конце сетки, что может произойти, когда количество элементов становится кратным ширине (другими словами, если не нужно добавлять заполняющие ячейки). Наконец, можно закрыть таблицу.

Например, напечатаем названия 50 штатов США в таблице из шести столбцов:

```
// устанавливаем соединение с базой данных
$dsn = 'mysql://user:password@localhost/table';
$dbh = DB::connect($dsn);
if (DB::isError($dbh)) { die ($dbh->getMessage()); }

// запрашиваем в базе данных информацию о 50-ти штатах
$sql = "SELECT state FROM states";
$sth = $dbh->query($sql);

// загружаем данные из базы данных в массив
while ($row = $sth->fetchRow(DB_FETCHMODE_ASSOC)) {
  $states[] = $row['state'];
}

// генерируем HTML-таблицу
$grid = pc_grid_horizontal($states, 6);

// и печатаем ее
print $grid;
```

При передаче в броузер это может выглядеть, как на рис. 4.1.

Рис. 4.1. *Соединенные Штаты Америки*

Поскольку 50 не делится без остатка на шесть, то в последней строке есть четыре дополнительных заполняющих ячейки.

5

Переменные

5.0. Введение

Вместе с условной логикой, переменные представляют то ядро, которое делает программы мощными и гибкими. Если вы представляете переменные как поименованные контейнеры, содержащие некое значение, то PHP допускает обычные контейнеры, контейнеры, хранящие имена других контейнеров, контейнеры с числами или строками, контейнеры, содержащие массивы других контейнеров, контейнеры с объектами и другие подобные варианты, которые только можно представить с помощью такой аналогии.

Переменная или установлена, или не установлена (сброшена). Переменная с любым присвоенным ей значением, `true` или `false`, пустым или не пустым, считается установленной. Функция `isset()` возвращает `true`, когда переданная ей переменная установлена. Единственным способом превращения установленной переменной в не установленную является вызов функции `unset()` для этой переменной. В функцию `unset()` можно передать скаляры, массивы и объекты. Можно также передать функции `unset()` несколько переменных, чтобы сбросить их все:

```
unset($vegetables);
unset($vegetables[12]);
unset($earth, $moon, $stars);
```

Если переменная присутствует в строке запроса URL, даже если ей не присвоили значение, то она установлена. Так:

```
http://www.example.com/set.php?chimps=&monkeys=12
```

устанавливает значение переменной `$_GET['monkeys']` равным `12`, а значение переменной `$_GET['chimps']` — равным пустой строке.

Все не установленные переменные также считаются пустыми. Установленные переменные могут быть пустыми или не пустыми. Пустые переменные имеют значения, которые оцениваются как логическое `false`: целое число 0; число с двойной точностью 0,0; пустая строка;

строка «0»; логическое false; массив без элементов; объект без переменных или методов и NULL. Все остальные значения считаются не пустыми. К ним относятся строка «00» и строка « », содержащая только символ пробела.

Переменные могут быть оценены или как true, или как false. Перечисленные ранее значения, приравненные к false, составляют все множество значений, которые приравнены к значению false в PHP. Все другие значения относятся к true. Разница между пустым значением и ложным состоит в том, что пустыми могут быть только переменные. Константы и значения, возвращаемые функциями, могут быть false, но не могут быть пустыми. Например, следующее выражение допустимо, поскольку $first_name представляет собой переменную:

```
if (empty($first_name)) { .. }
```

Напротив, следующие два выражения порождают синтаксические ошибки из-за 0 (константа) и значения, возвращаемого функцией get_first_name(), которые не могут быть пустыми:

```
if (empty(0)) { .. }
if (empty(get_first_name())) { .. }
```

5.1. Операторы == и =: как избежать путаницы

Задача

Необходимо избежать случайного присваивания значения при сравнении переменной с константой.

Решение

Запись, представленную ниже:

```
if (12 == $dwarves) { ... }
```

следует предпочесть такой:

```
if ($dwarves == 12) { ... }
```

Если константу расположить слева, то использование оператора присваивания вызовет синтаксическую ошибку. Другими словами, PHP выразит недовольство, если написать:

```
if (12 = $dwarves) { ... }
```

но код:

```
if ($dwarves = 12) { ... }
```

выполнит молча, сначала присвоив значение 12 переменной $dwarves, а затем отработает код внутри блока. (Выражение $dwarves = 12 приравнивается к 12, что рассматривается как true.)

Обсуждение

Размещение константы слева от оператора сравнения приводит результат сравнения к типу константы. Это может вызвать ошибку при сравнении целого числа с переменной, которая может быть целым числом или строкой. Выражение `0 == $dwarves` имеет значение `true`, когда переменная `$dwarves` равна `0`, но оно также истинно, когда `$dwarves` содержит строку `sleepy`. Целое число (`0`) находится слева от оператора сравнения, поэтому перед сравнением PHP преобразует правую часть (строку `sleepy`) в целое число (`0`). Во избежание этого вместо оператора сравнения следует применять оператор тождества `0 === $dwarves`.

См. также

Документацию по оператору `=` на *http://www.php.net/language.operators.assignment.php* и по операторам `==` и `===` на *http://www.php.net/manual/language.operators.comparison.php*.

5.2. Установка значения по умолчанию

Задача

Необходимо присвоить значение по умолчанию переменной, у которой еще нет значения. Часто бывает необходимо присвоить переменной жестко запрограммированное значение по умолчанию, которое может быть перезаписано значением, введенным пользователем в форме, или значением переменной окружения.

Решение

Значение по умолчанию переменной, которая, возможно, уже имеет значение, присваивается при помощи функции `isset()`:

```
if (! isset($cars)) { $cars = $default_cars; }
```

А трехчленный оператор (`a ? b : c`) позволяет присвоить значение (возможно, значение по умолчанию) новой переменной:

```
$cars = isset($_REQUEST['cars']) ? $_REQUEST['cars'] : $default_cars;
```

Обсуждение

Применение функции `isset()` имеет важнейшее значение в случае присваивания значений по умолчанию. Без нее значение не по умолчанию не может быть равным `0` или чему бы то ни было еще, что приравнивается к `false`. Рассмотрим следующее присваивание:

```
$cars = $_REQUEST['cars'] ? $_REQUEST['cars'] : $default_cars;
```

Если `$REQUEST['cars']` равно `0`, то `$cars` устанавливается в `$default_cars`, даже если `0` является допустимым значением для `$cars`.

Для упрощения присваивания множества значений по умолчанию можно использовать массив таких значений. Ключи в этом массиве представляют имена переменных, а значения массива – это значения по умолчанию для каждой из переменных:

```
$defaults = array('emperors'  => array('Rudolf II','Caligula'),
                  'vegetable' => 'celery',
                  'acres'     => 15);

foreach ($defaults as $k => $v) {
    if (! isset($GLOBALS[$k])) { $GLOBALS[$k] = $v; }
}
```

Переменные находятся в глобальном пространстве имен, поэтому предыдущий код не может применяться для установки локальных значений внутри функций. Для этого нужны переменные переменных:

```
foreach ($defaults as $k => $v) {
    if (! isset($$k)) { $$k = $v; }
}
```

См. также

Документацию по функции `isset()` на *http://www.php.net/isset*; переменные переменных обсуждаются в рецепте 5.4 и на *http://www.php.net/language.variables.variable*.

5.3. Обмен значениями без временных переменных

Задача

Необходимо взаимно обменять значения двух переменных без использования дополнительной переменной для промежуточного хранения значений.

Решение

Взаимно обменять `$a` и `$b` можно так:

```
list($a,$b) = array($b,$a);
```

Обсуждение

Конструкция языка PHP `list()` позволяет присваивать значения из массива отдельным переменным. Ее двойник, функция `array()`, стоящая в правой части выражения, позволяет конструировать массивы из отдельных переменных. Присваивание массива, возвращенного функцией `array()`, переменным в функции `list()` позволяет жонглировать порядком этих значений. Этот способ подходит и для более чем двух переменных, например:

```
list($yesterday,$today,$tomorrow) = array($today,$tomorrow,$yesterday);
```

Данный способ не дает преимущества в скорости по сравнению с временными переменными, поэтому его применяют для прозрачности кода, а не ради скорости.

См. также

Документацию по функции `list()` на *http://www.php.net/list* и по функции `array()` на *http://www.php.net/ array*.

5.4. Создание динамического имени переменной

Задача

Необходимо создавать имя переменной динамически. Например, требуется дать переменным имена, совпадающие с именами полей в запросе к базе данных.

Решение

В PHP для применения синтаксиса переменных переменных в начало переменной, значение которой является требуемым именем переменной, добавляется символ `$`:

```
$animal = 'turtles';
$turtles = 103;
print $$animal;
103
```

Обсуждение

Код предыдущего примера печатает `103`. Так как `$animal = 'turtles'`, то переменная `$$animal` равна `$turtles`, которая, в свою очередь, равна 103.

Фигурные скобки позволяют построить более сложные выражения, обозначающие имена переменных:

```
$stooges = array('Moe','Larry','Curly');
$stooge_moe = 'Moses Horwitz';
$stooge_larry = 'Louis Feinberg';
$stooge_curly = 'Jerome Horwitz';

foreach ($stooges as $s) {
  print "$s's real name was ${'stooge_'.strtolower($s)}.\n";
}
Moe's real name was Moses Horwitz.
Larry's real name was Louis Feinberg.
Curly's real name was Jerome Horwitz.
```

PHP вычисляет выражение, заключенное в фигурные скобки, и использует его в качестве имени переменной. Это выражение может даже включать в себя вызовы функций, например `strtolower()`.

Переменные переменные удобны также для выполнения итераций по переменным, имеющим похожие имена. Скажем, из базы данных запрашивается таблица, поля которой имеют имена `title_1`, `title_2` и т. д. Если требуется проверить, совпадает ли заголовок с одним из этих имен, то проще всего выполнить цикл по этим именам, примерно так:

```
for ($i = 1; $i <= $n; $i++) {
    $t = "title_$i";
    if ($title == $$t) { /* совпадение */ }
}
```

Несомненно, естественнее хранить значения в массиве, но если вы поддерживаете старый код, в котором применяются эти приемы (и этот код нельзя изменить), то переменные переменные будут полезны.

Синтаксис фигурных скобок также необходим для разрешения неопределенностей в элементах массива. Переменная переменная `$$donkeys[12]` может иметь два значения. Первое значение: «Возьмите то, что находится в 12-м элементе массива `$donkeys`, и используйте в качестве имени переменной». Записывается это так: `${$donkeys[12]}`. Второе значение: «Используйте содержимое скаляра `$donkeys` в качестве имени массива и загляните в 12-й элемент этого массива». Запись: `${$donkeys}[12]`.

См. также

http://www.php.net/language.variables.variable, где находится документация по переменным переменным.

5.5. Статические переменные

Задача

Необходима локальная переменная для сохранения значений между вызовами функции.

Решение

Объявите переменную как статическую:

```
function track_times_called() {
    static $i = 0;
    $i++;
    return $i;
}
```

Обсуждение

Функция запоминает переменную, объявленную как статическую. Поэтому при последовательных вызовах функций можно получить доступ

к значению сохраненной переменной. Функция `pc_check_the_count()`, показанная в примере 5.1, содержит статическую переменную, позволяющую отслеживать удары и мячи для отбивающего в бейсболе.

Пример 5.1. pc_check_the_count( )

```
function pc_check_the_count($pitch) {
    static $strikes = 0;
    static $balls   = 0;

    switch ($pitch) {
    case 'foul':
        if (2 == $strikes) break;        // при двух ударах
                                            ничего не происходит
        // в противном случае действует, как удар
    case 'strike':
        $strikes++;
        break;
    case 'ball':
        $balls++;
        break;
    }

    if (3 == $strikes) {
        $strikes = $balls = 0;
        return 'strike out';
    }
    if (4 == $balls) {
        $strikes = $balls = 0;
        return 'walk';
    }
    return 'at bat';
}

$what_happened = check_the_count($pitch);
```

Логика происходящего с отбивающим, зависящая от количества подач, содержится в операторе `switch` внутри функции `pc_check_the_count()`. Вместо этого можно вернуть количество ударов и мячей, пробежек или простоев, но тогда надо добавить соответствующий код проверки на отражения ударов, пробежки и простои.

Несмотря на то что статические переменные хранят свои значения все время между вызовами функций, они делают это только в течение одного вызова сценария. Статическая переменная, полученная в результате одного запроса, не сохранит свое значение для следующего запроса той же самой страницы.

См. также

Документацию по статическим переменным на *http://www.php.net/language.variables.scope*.

5.6. Совместное использование переменных процессами

Задача

Необходимо найти способ совместного использования информации несколькими процессами, что обеспечивает более быстрый доступ к разделяемому ресурсу.

Решение

Занесите данные в совместно используемый сегмент памяти и обеспечьте эксклюзивный доступ к разделяемому ресурсу с помощью семафора:

```
$semaphore_id = 100;
$segment_id   = 200;
// захватываем дескриптор семафора, связанного с нужным сегментом памяти
$sem = sem_get($semaphore_id,1,0600);
// обеспечиваем эксклюзивный доступ к семафору
sem_acquire($sem) or die("Can't acquire semaphore");
// определяем дескриптор для разделяемого сегмента памяти
$shm = shm_attach($segment_id,16384,0600);
// возвращаем значение из разделяемого сегмента памяти
$population = shm_get_var($shm,'population');
// обрабатываем значение
$population += ($births + $immigrants - $deaths - $emigrants);
// заносим значение обратно в разделяемый сегмент памяти
shm_put_var($shm,'population',$population);
// освобождаем дескриптор разделяемого сегмента памяти
shm_detach($shm);
// освобождаем дескриптор семафора,
// так чтобы другой процесс мог его получить
sem_release($sem);
```

Обсуждение

Разделяемый сегмент памяти – это часть оперативной памяти машины, к которой могут получить доступ различные процессы (такие как разнообразные веб-серверы, обрабатывающие запросы). Семафор помогает процессам не путаться друг у друга под ногами, когда они получают доступ к разделяемому сегменту памяти. Прежде чем процесс сможет использовать сегмент, он должен получить управление семафором. По окончании работы с сегментом он освобождает семафор, чтобы другой процесс имел возможность его захватить.

Получить контроль над семафором позволяет функция `sem_get()`, которая определяет его идентификатор. Первым аргументом функции `sem_get()` является целочисленный ключ семафора. В качестве ключа может выступать любое целое число, одинаковое для всех программ, которым требуется доступ к определенному семафору. Если семафор

с указанным ключом еще не существует, то он создается; второй аргумент (в данном случае 1) устанавливает максимальное количество процессов, которые могут получить доступ к семафору, а права доступа к семафору устанавливаются третьим аргументом (0600) функции sem_get(). Эти права доступа работают так же, как и права доступа к файлу, поэтому 0600 означает, что пользователь, создавший семафор, может читать и записывать их. В этом контексте «пользователь» означает не только процесс, создавший семафор, но и любой процесс с тем же самым идентификатором пользователя. Права доступа 0600 должны подойти в большинстве случаев, в которых процессы веб-сервера запускаются от имени одного и того же пользователя.

Функция sem_get() возвращает идентификатор, указывающий на базовый системный семафор. Этот идентификатор позволяет получить контроль над семафором с помощью функции sem_acquire(). Эта функция ожидает возможности захвата семафора (возможно, ожидает, пока другой процесс не освободит семафор) и затем возвращает значение true. В случае ошибки она возвращает false. К ошибкам относятся неверные права доступа или недостаток памяти для создания семафора. Как только семафор захвачен, можно читать из разделяемого сегмента памяти.

Сначала устанавливаем ссылку на выбранный разделяемый сегмент памяти с помощью функции shm_attach(). Как и в функции sem_get(), первым аргументом функции shm_attach() является целочисленный ключ. Однако на этот раз он идентифицирует требуемый сегмент, а не семафор. Если сегмент с указанным ключом не существует, то остальные аргументы его создают. Второй аргумент (16384) представляет размер сегмента в байтах, а последний аргумент задает права доступа к сегменту. Функция shm_attach(200,16384,0600) создает разделяемый сегмент памяти размером 16 Кбайт, читать из которого и записывать в который может только пользователь, его создавший. Эта функция возвращает идентификатор, необходимый для чтения и записи в разделяемый сегмент памяти.

После подсоединения к сегменту переменные из него извлекают с помощью функции shm_get_var($shm, 'population'). Она заглядывает в разделяемый сегмент памяти, определяемый переменной $shm, и извлекает значение переменной под именем population. В разделяемой памяти можно хранить переменные любого типа. С извлеченной переменной можно работать так же, как и с другими переменными. Функция shm_put_var($shm,'population',$population) помещает значение переменной $population обратно в разделяемый сегмент памяти в виде переменной под именем population.

Теперь работа с разделяемым сегментом памяти закончена. Отсоединитесь от него с помощью функции shm_detach() и освободите семафор с помощью функции sem_release(), чтобы другой процесс смог его использовать.

Основное преимущество разделяемой памяти заключается в ее скорости. Но так как она располагается в оперативной памяти, то не может хранить слишком большие данные и не сохраняется во время перезагрузки машины (если не принять специальных мер по записи информации, хранящейся в разделяемой памяти, на диск и обратной ее записи после перезапуска компьютера). Кроме того, разделяемая память не поддерживается в Windows.

См. также

Рецепт 8.27, содержащий программу, работающую с разделяемой памятью; документацию по разделяемой памяти и семафорным функциям на *http://www.php.net/sem*.

5.7. Сериализация данных сложных типов в виде строки

Задача

Необходимо строковое представление массива или объекта для занесения в файл или базу данных. Требуется обеспечить легкость обратного преобразования строки в массив или объект.

Решение

Для преобразования переменных и их значений в текстовую форму применяется функция `serialize()`:

```
$pantry = array('sugar' => '2 lbs.','butter' => '3 sticks');
$fp = fopen('/tmp/pantry','w') or die ("Can't open pantry");
fputs($fp,serialize($pantry));
fclose($fp);
```

Для воссоздания переменных предназначена функция `unserialize()`:

```
$new_pantry = unserialize(join('',file('/tmp/pantry')));
```

Обсуждение

Сформированная строка, преобразуемая обратно в массив `$pantry`, выглядит следующим образом:

```
a:2:{s:5:"sugar";s:6:"2 lbs.";s:6:"butter";s:8:"3 sticks";}
```

Здесь достаточно информации, позволяющей перевести все значения обратно в массив, но само имя переменной в последовательном представлении не сохраняется.

К данным, передаваемым в последовательной форме со страницы на страницу с помощью URL, необходимо применять функцию `urlencode()` для преобразования метасимволов URL в escape-последовательности:

```
$shopping_cart = array('Poppy Seed Bagel' => 2,
                       'Plain Bagel' => 1,
                       'Lox' => 4);
print '<a
href="next.php?cart='.urlencode(serialize($shopping_cart)).'">Next</a>';
```

На передаваемые в функцию `unserialize()` данные оказывают влияние параметры настройки `magic_quotes_gpc` и `magic_quotes_runtime`. Если параметр `magic_quotes_gpc` равен `on`, то данные, передаваемые в URL, POST-переменные или cookies должны быть обработаны с помощью функции `stripslashes()` перед преобразованием в последовательную форму:

```
$new_cart = unserialize(stripslashes($cart));
                                  // если параметр magic_quotes_gpc равен on
$new_cart = unserialize($cart);   // если параметр magic_quotes_gpc равен off
```

Если параметр `magic_quotes_runtime` равен `on`, то данные в последовательной форме, сохраняемые в файле, необходимо обрабатывать при записи с помощью функции `addslashes()`, а при чтении – с помощью функции `stripslashes()`:

```
$fp = fopen('/tmp/cart,'w');
fputs($fp,addslashes(serialize($a)));
fclose($fp);

// если параметр magic_quotes_runtime равен on
$new_cart = unserialize(stripslashes(join('',file('/tmp/cart'))));
// если параметр magic_quotes_runtime равен off
$new_cart = unserialize(join('',file('/tmp/cart')));
```

Данные в последовательной форме, прочитанные из базы данных, также должны быть переданы функции `stripslashes()`, если параметр `magic_quotes_runtime` равен `on`:

```
mysql_query(
    "INSERT INTO cart (id,data) VALUES (1,'".addslashes(serialize($cart))."')");

$r = mysql_query('SELECT data FROM cart WHERE id = 1');
$ob = mysql_fetch_object($r);
// если параметр magic_quotes_runtime равен on
$new_cart = unserialize(stripslashes($ob->data));
// если параметр magic_quotes_runtime равен off
$new_cart = unserialize($ob->data);
```

Данные в последовательной форме, передаваемые в базу данных, также должны обрабатываться функцией `addslashes()` (или должен применяться другой, подходящий для базы данных метод escape-кодировки) для их корректного сохранения.

См. также

Рецепт 10.7 об escape-кодировке информации, заносимой в базу данных.

5.8. Получение дампа содержимого переменных в виде строк

Задача

Необходимо проверить значения, хранимые в переменных. Это может быть вложенный массив или объект, поэтому нельзя просто распечатать его и пройтись по нему в цикле.

Решение

Для этого следует применять функцию `print_r()` или функцию `var_dump()`:

```
$array = array("name" => "frank", 12, array(3, 4));

print_r($array);
Array
(
    [name] => frank
    [0] => 12
    [1] => Array
        (
            [0] => 3
            [1] => 4
        )
)
var_dump($array);
array(3) {
  ["name"]=>
  string(5) "frank"
  [0]=>
  int(12)
  [1]=>
  array(2) {
    [0]=>
    int(3)
    [1]=>
    int(4)
  }
}
```

Обсуждение

Вывод функции `print_r()` короче и его легче читать. Однако вывод функции `var_dump()` содержит типы данных и длину каждой переменной.

Эти функции работают с переменными рекурсивно, поэтому если внутри переменной есть ссылки на саму себя, то в результате можно получить бесконечный цикл. Хотя обе функции сами умеют избегать бесконечного вывода значений переменных. Функция `print_r()` после перво-

го вхождения переменной печатает слово *RECURSION* вместо дальнейшего вывода информации об этой переменной и продолжает итерацию для оставшихся данных, которые она должна вывести на печать. Если функция var_dump() встречает переменную более трех раз, она выдает фатальную ошибку и заканчивает выполнение сценария. Рассмотрим массивы $user_1 и $user_2, ссылающиеся друг на друга посредством элементов friend:

```
$user_1 = array('name' => 'Max Bialystock',
                'username' => 'max');

$user_2 = array('name' => 'Leo Bloom',
                'username' => 'leo');

// Макс и Лео - друзья
$user_2['friend'] = &$user_1;
$user_1['friend'] = &$user_2;

// у Макса и Лео есть работа
$user_1['job'] = 'Swindler';
$user_2['job'] = 'Accountant';
```

Вывод функции print_r($user_2):

```
Array
(
    [name] => Leo Bloom
    [username] => leo
    [friend] => Array
        (
            [name] => Max Bialystock
            [username] => max
            [friend] => Array
                (
                    [name] => Leo Bloom
                    [username] => leo
                    [friend] => Array
 *RECURSION*
                    [job] => Accountant
                )

            [job] => Swindler
        )

    [job] => Accountant
)
```

Встретив ссылку на $user_1 второй раз, функция print_r() печатает слово *RECURSION* вместо обращения к массиву. Затем она продолжает свою работу, печатая оставшиеся элементы массивов $user_1 и $user_2.

Встретившись с рекурсией, функция var_dump() ведет себя по-другому:

```
array(4) {
  ["name"]=>
```

```
    string(9) "Leo Bloom"
    ["username"]=>
    string(3) "leo"
    ["friend"]=>
    &array(4) {
      ["name"]=>
      string(14) "Max Bialystock"
      ["username"]=>
      string(3) "max"
      ["friend"]=>
      &array(4) {
        ["name"]=>
        string(9) "Leo Bloom"
        ["username"]=>
        string(3) "leo"
        ["friend"]=>
        &array(4) {
          ["name"]=>
          string(14) "Max Bialystock"
          ["username"]=>
          string(3) "max"
          ["friend"]=>
          &array(4) {
            ["name"]=>
            string(9) "Leo Bloom"
            ["username"]=>
            string(3) "leo"
            ["friend"]=>
            &array(4) {
              ["name"]=>
              string(14) "Max Bialystock"
              ["username"]=>
              string(3) "max"
              ["friend"]=>
              &array(4) {
                ["name"]=>
                string(9) "Leo Bloom"
                ["username"]=>
                string(3) "leo"
                ["friend"]=>
                &array(4) {
<br />
<b>Fatal error</b>:  Nesting level too deep - recursive dependency? in
<b>var-dump.php</b> on line <b>15</b><br />
```

Функция `var_dump()` останавливает рекурсию еще до четвертого появления ссылки на массив `$user_1`. Когда это происходит, она выдает фатальную ошибку и прекращает выдачу дампа переменных (или выполнение сценария).

И хотя функции `print_r()` и `var_dump()` печатают свои результаты вместо того, чтобы их возвратить, они могут сохранить данные без их распечатки, используя выходной буфер:

```
ob_start();
var_dump($user);
$dump = ob_get_contents();
ob_end_clean();
```

Таким образом, результаты функции `var_dump($user)` помещаются в переменную `$dump`.

См. также

Буферизация вывода обсуждается в рецепте 8.12; обработка ошибок с помощью модуля PEAR DB, показанная в рецепте 10.8, основана на буферизации вывода функции `print_r()` для сохранения сообщений об ошибках; документацию по функции `print_r()` на *http://www.php.net/print-r* и по функции `var_dump()` на *http://www.php.net/var-dump*.

6

Функции

6.0. Введение

Функции помогают разрабатывать структурированный код, который можно использовать неоднократно. Они позволяют скрыть детали, так что код становится более гибким и легко читаемым. Без функций невозможно написать легко поддерживаемые программы из-за необходимости непрерывно обновлять одни и те же блоки кода во многих местах и во многих файлах.

Работая с функцией, вы передаете ей некоторое количество аргументов и получаете обратно значение:

```
// складываем два числа
function add($a, $b) {
    return $a + $b;
}

$total = add(2, 2);    // 4
```

Функция объявляется при помощи ключевого слова `function`, за которым следуют ее имя и какие-либо параметры в скобках. Для вызова функции достаточно указать ее имя и задать значения аргументов для каждого ее параметра. Если функция возвращает значение, то можно присвоить результат функции переменной, как показано в предыдущем примере.

Не обязательно предварительно объявлять функцию перед ее вызовом. PHP анализирует весь файл до его выполнения, поэтому можно произвольно размещать объявления функций и их вызовы. Однако в PHP запрещено переопределение функций. Если PHP обнаруживает функцию с именем ранее найденной функции, то он выдает фатальную ошибку и «умирает».

Иногда стандартная процедура выполнения с фиксированным числом аргументов и одним возвращаемым значением не вполне подходит для конкретной ситуации в коде. Возможно, заранее не известно количество параметров, которые функция должна принимать. Или о парамет-

рах все известно, но они почти всегда имеют те же самые значения, и каждый раз снова передавать их довольно утомительно. Или необходимо, чтобы функция возвращала более одного значения.

Эта глава поможет вам разрешать проблемы такого рода с помощью PHP. Мы начнем с подробного рассмотрения различных способов передачи аргументов в функцию. Рецепты с 6.1 по 6.5 охватывают передачу аргументов по значению, по ссылке и как именованных параметров; присваивание параметрам значений по умолчанию; функции с переменным количеством параметров.

Следующие четыре рецепта полностью посвящены возвращению значений из функции. Рецепт 6.6 рассматривает возвращение значения по ссылке, рецепт 6.7 охватывает возвращение более одной переменной, рецепт 6.8 описывает, как пропускать определенные возвращаемые значения, а рецепт 6.9 рассказывает о наилучшем способе получения информации об ошибках в работе функции и ее проверки. Последние три рецепта показывают, как вызывать переменные функции, решать проблемы переменных областей видимости и динамически создавать функцию. В главе 5 есть рецепт о переменных функций; о сохранении значений переменных между вызовами функций рассказано в рецепте 5.5.

6.1. Доступ к параметрам функций

Задача

Необходимо получить доступ к значениям, переданным в функцию.

Решение

Этот доступ можно получить посредством имен из прототипа функции:

```
function commercial_sponsorship($letter, $number) {
    print "This episode of Sesame Street is brought to you by ";
    print "the letter $letter and number $number.\n";
}

commercial_sponsorship('G', 3);
commercial_sponsorship($another_letter, $another_number);
```

Обсуждение

Внутри функции не имеет значения, как в нее передавались параметры: в виде строк, чисел, массивов или переменных других типов. Можно считать, что они имеют те же самые типы, и ссылаться на них по именам из прототипа.

В противоположность языку C, нет необходимости (да в действительности, и возможности) описывать тип передаваемой переменной. PHP следит за этим за вас.

Кроме того, если не определено обратное, все значения передаются в функцию и из функции по значению, а не по ссылке. Это означает, что PHP создает копию значения и предоставляет эту копию для обработки. Поэтому любые изменения, происходящие с копией, не влияют на оригинальное значение. Например:

```
function add_one($number) {
    $number++;
}

$number = 1;
add_one($number);
print "$number\n";
1
```

Если бы переменную передали по ссылке, то значение переменной `$number` было бы равно 2.

Во многих языках передача переменных по ссылке имеет еще одно преимущество – она значительно быстрее передачи по значению. И хотя для PHP это тоже справедливо, но разница в скорости минимальная. По этой причине мы предлагаем прибегать к передаче по ссылке только при реальной необходимости и никогда в целях улучшения производительности.

См. также

Рецепт 6.3 о передаче значений по ссылке и рецепт 6.6 о возвращении значений по ссылке.

6.2. Установка значений по умолчанию для параметров функции

Задача

Необходимо, чтобы параметр получил значение по умолчанию, если вызывающий функцию не передал для него никакого значения. Например, если в функцию вывода таблицы не передается значение ширины таблицы, то по умолчанию для этого параметра может быть установлено значение, равное 1.

Решение

Присвойте параметру значение по умолчанию в прототипе функции:

```
function wrap_html_tag($string, $tag = 'b') {
    return "<$tag>$string</$tag>";
}
```

Обсуждение

Пример в разделе «Решение» устанавливает для полужирного текста значение по умолчанию, равное `b`. Например:

```
$string = 'I am some HTML';
wrap_html_tag($string);
```

возвращает:

```
<b>I am some HTML</b>
```

Следующий пример:

```
wrap_html_tag($string, 'i');
```

возвращает:

```
<i>I am some HTML</i>
```

Определяя значения по умолчанию, необходимо помнить две важные вещи. Во-первых, все параметры со значениями по умолчанию должны следовать за параметрами без значений по умолчанию. В противном случае PHP не сможет определить, какие параметры опущены и должны принять значения по умолчанию, а какие аргументы заменяют значение по умолчанию. Поэтому функция `wrap_html_tag()` не может быть определена следующим образом:

```
function wrap_html_tag($tag = 'i', $string)
```

Если так сделать и передать функции `wrap_html_tag()` только один аргумент, то PHP присвоит это значение переменной `$tag` и выдаст предупреждение, выражая недовольство пропуском второго аргумента.

Во-вторых, присвоенное значение должно быть константой: строкой или числом. Оно не может быть переменной. В качестве примера опять возьмем функцию `wrap_html_tag()`. Так делать нельзя:

```
$my_favorite_html_tag = 'i';

function wrap_html_tag($string, $tag = $my_favorite_html_tag) {
    ...
}
```

Если необходимо, чтобы по умолчанию не было присвоено ничего, то единственный способ это сделать состоит в том, чтобы присвоить параметру пустую строку:

```
function wrap_html_tag($string, $tag = '') {
    if (empty($tag)) return $string;
    return "<$tag>$string</$tag>";
}
```

Эта функция возвращает оригинальную строку, если для переменной `$tag` не было передано никакого значения. А если был передан тег (не пустой), то она возвратит строку, находящуюся внутри тегов.

В зависимости от обстоятельств другой альтернативой значения по умолчанию для переменной `$tag` являются значения `0` или `NULL`. В функции `wrap_html_tag()` пустое значение тега нам ни к чему. Однако в некоторых случаях пустая строка может быть вполне допустима. Например, функция `join()` часто вызывается с пустой строкой после вызова функции `file()`, чтобы поместить файл в строку. Кроме того, как показывает следующий код, можно использовать сообщение по умолчанию, если не передано никаких аргументов, и пустое сообщение, если передана пустая строка:

```
function pc_log_db_error($message = NULL) {
    if (is_null($message)) {
        $message = 'Couldn't connect to DB';
    }

    error_log("[DB] [$message]");
}
```

См. также

Рецепт 6.5, где рассказано о создании функций, принимающих переменное количество аргументов.

6.3. Передача значений по ссылке

Задача

Необходимо передать переменную в функцию, так чтобы эта переменная сохраняла любые изменения, происходящие с ее значением внутри функции.

Решение

Для того чтобы функция принимала аргументы по ссылке, а не по значению, поставьте символ `&` перед именем параметра в прототипе функции:

```
function wrap_html_tag(&$string, $tag = 'b') {
    $string = "<$tag>$string</$tag>";
}
```

Теперь нет необходимости возвращать строку, поскольку изменяется сам оригинал.

Обсуждение

Передача переменной в функцию по ссылке позволяет избежать работы по возвращению переменной и присваиванию возвращенного значения исходной переменной. Это также удобно, если вы хотите, чтобы функция возвращала в случае успеха или неудачи логические значения `true` или `false`, но в то же время она могла бы еще модифицировать значения аргумента.

Можно выбирать между передачей параметра по ссылке или по значению – либо одно, либо другое. Другими словами, нельзя заставить PHP произвольно выбирать между передачей переменной по ссылке или по значению.

В действительности это утверждение не на 100% правда. Если конфигурационная директива `allow_call_time_pass_reference` разрешена, то PHP разрешает не передавать значение по ссылке, если символ амперсанда предшествует имени переменной. Однако начиная с версии PHP 4.0 Beta 4 использование этой возможности не приветствуется, и PHP предупреждает, что эта функциональность в будущем может быть исключена при использовании вызовов с передачей параметров по ссылке. Программистам следует быть внимательными.

Кроме того, если параметр объявлен для передачи по ссылке, то нельзя передавать строку (или число), иначе PHP завершит работу с фатальной ошибкой.

См. также

Рецепт 6.6, в котором описано возвращение значений по ссылке.

6.4. Именованные параметры

Задача

Необходимо задавать аргументы функции по имени, а не просто по их местоположению в вызове функции.

Решение

Определите функцию с одним параметром, но сделайте его ассоциативным массивом:

```
function image($img) {
    $tag  = '<img src="' . $img['src'] . '" ';
    $tag .= 'alt="' . ($img['alt'] ? $img['alt'] : '') .'">';
    return $tag;
}

$image = image(array('src' => 'cow.png', 'alt' => 'cows say moo'));
$image = image(array('src' => 'pig.jpeg'));
```

Обсуждение

Применение именованных параметров усложняет внутренний код функций, но облегчает чтение вызывающего кода. Функция располагается в одном месте, а вызывается из многих, что делает код более понятным.

Если при этом допустить ошибку в имени параметра, то PHP на это не отреагирует, поэтому нужна аккуратность, поскольку анализатор ко-

да такую ошибку не обнаружит. Кроме того, становится недоступным такое преимущество PHP, как возможность присваивать параметру значения по умолчанию. К счастью, этот недостаток можно обойти с помощью простого кода в начале функции:

```
function image($img) {
    if (! isset($img['src']))    { $img['src']    = 'cow.png';      }
    if (! isset($img['alt']))    { $img['alt']    = 'milk factory'; }
    if (! isset($img['height])) { $img['height'] = 100;            }
    if (! isset($img['width']))  { $img['width']  = 50;             }
    ...
}
```

Работая с функцией `isset()`, проверьте, установлено ли значение для каждого из параметров, и если нет, то присвойте ему значение по умолчанию.

Есть и альтернативный вариант – напишите короткую функцию, которая будет это делать:

```
function pc_assign_defaults($array, $defaults) {
    $a = array();
    foreach ($defaults as $d => $v) {
        $a[$d] = isset($array[$d]) ? $array[$d] : $v;
    }

    return $a;
}
```

Эта функция выполняет цикл по последовательности ключей массива значений по умолчанию и проверяет, содержит ли данный массив `$array` множество значений. Если нет, то функция присваивает значения из массива `$defaults`. Чтобы использовать ее в предыдущем фрагменте, замените верхние строки на:

```
function image($img) {
    $defaults = array('src'    => 'cow.png',
                      'alt'    => 'milk factory',
                      'height' => 100,
                      'width'  => 50
                     );
    $img = pc_assign_defaults($img, $defaults);
    ...
}
```

Это выглядит лучше, поскольку придает коду большую гибкость. Если необходимо модифицировать присваивание значений по умолчанию, то достаточно внести изменения в код функции `pc_assign_defaults()`, а не в сотнях строк кода различных функций. Кроме того, легче иметь массив пар имя/значение и одну строку, которая присваивает значения по умолчанию, чем смешивать две концепции в серии почти одинаковых повторяющихся строк.

См. также

Рецепт 6.5, описывающий создание функции, принимающей переменное количество аргументов.

6.5. Создание функции, принимающей переменное количество аргументов

Задача

Необходимо определить функцию, принимающую переменное количество аргументов.

Решение

Передайте массив и поместите в него переменные аргументы:

```
// определение среднего группы чисел
function mean($numbers) {
    // инициализируем, чтобы избежать предупреждений
    $sum = 0;

    // количество элементов в массиве
    $size = count($numbers);

    // выполняем цикл по массиву и суммируем числа
    for ($i = 0; $i < $size; $i++) {
        $sum += $numbers[$i];
    }

    // делим на количество чисел
    $average = $sum / $size;

    // возвращаем среднее
    return $average;
}

$mean = mean(array(96, 93, 97));
```

Обсуждение

Есть два хороших решения, зависящих от стиля программирования и предпочтений программиста. Более традиционным для PHP является метод, описанный выше в разделе «Решение». Мы предпочитаем именно его, т. к. применение массивов в PHP – обычное дело, и все программисты хорошо знакомы с массивами и их поведением.

Таким образом, хотя этот метод требует некоторых дополнительных накладных расходов, группирование переменных общепринято. Оно применяется в рецепте 6.4 для создания именованных параметров и в рецепте 6.7 для возвращения из функции более одного значения. Кроме того, внутри функции синтаксис доступа и манипуляции эле-

ментами массива включает такие основные команды, как `$array[$i]` и `count($array)`.

Однако выглядеть это может неуклюже, поэтому PHP обеспечивает альтернативу и разрешает прямой доступ к списку аргументов:

```
// определение среднего группы чисел
function mean() {
    // инициализируем, чтобы избежать предупреждений
    $sum = 0;

    // количество аргументов, переданных в функцию
    $size = func_num_args();

    // выполняем цикл по аргументам и суммируем числа
    for ($i = 0; $i < $size; $i++) {
        $sum += func_get_arg($i);
    }

    // делим на количество чисел
    $average = $sum / $size;

    // возвращаем среднее
    return $average;
}

$mean = mean(96, 93, 97);
```

В этом примере задействован ряд функций, возвращающих данные, основанные на аргументах, переданных функции, из которой они вызываются. Сначала функция `func_num_args()` возвращает целое число, показывающее количество аргументов, переданных в вызывающую ее функцию; в данном случае это функция `mean()`. Затем отсюда можно вызвать функцию `func_get_arg()`, чтобы определить конкретное значение аргумента для каждой позиции.

При вызове функции `mean(96, 93, 97)` функция `func_num_args()` возвращает `3`. Первый аргумент находится в позиции `0`, поэтому цикл выполняется от `0` до `2`, а не от `1` до `3`. То есть это происходит в цикле `for`, когда переменная `$i` пробегает значения от `0` до числа, меньшего `$size`. Как можно видеть, это та же самая логика, что была реализована в первом примере, в котором был передан массив. Можно не беспокоиться о возможных накладных расходах от вызова функции `func_get_arg()` внутри цикла. Эта версия в действительности быстрее метода передачи массива.

Ниже приведена третья версия этой функции, в которой функция `func_num_args()` возвращает массив, содержащий все значения, переданные функции. Ее завершение выглядит как гибрид двух предыдущих функций:

```
// определение среднего группы чисел
function mean() {
    // инициализируем, чтобы избежать предупреждений
    $sum = 0;
```

```
    // загружаем аргументы в массив $numbers
    $numbers = func_get_args();

    // количество элементов в массиве
    $size = count($numbers);

    // выполняем цикл по массиву и суммируем числа
    for ($i = 0; $i < $size; $i++) {
        $sum += $numbers[$i];
    }

    // делим на количество чисел
    $average = $sum / $size;

    // возвращаем среднее
    return $average;
}

$mean = mean(96, 93, 97);
```

Здесь мы получаем двойную выгоду от того, что нет необходимости помещать числа во временный массив для передачи их в функцию `mean()`, но внутри функции можно трактовать их так, как будто это сделано. К сожалению, этот способ несколько медленнее первых двух.

См. также

Рецепт 6.7 о возвращении множества значений из функции; документацию по функции `func_num_arg()` на *http://www.php.net/func-num-arg*, по функции `func_get_arg()` на *http://www.php.net/func-get-arg* и по функции `func_get_args()` на *http://www.php.net/func-get-args*.

6.6. Возвращение значений по ссылке

Задача

Необходимо вернуть значение по ссылке, а не по значению. Это позволяет избежать создания еще одной копии переменной.

Решение

Синтаксис возвращения переменной по ссылке подобен синтаксису передачи ее по ссылке. Однако вместо размещения символа `&` перед параметром располагаем его перед именем функции:

```
function &wrap_html_tag($string, $tag = 'b') {
    return "<$tag>$string</$tag>";
}
```

Кроме того, при вызове функции нужно использовать оператор присваивания `=&`, а не обычный оператор `=`:

```
$html =& wrap_html_tag($string);
```

Обсуждение

В отличие от передачи значения в функцию, когда аргумент передается либо по значению, либо по ссылке, в данном случае не обязательно выбирать присваивание ссылки, а можно просто взять возвращенное значение. Достаточно заменить обычным оператором `=` оператор `=&`, и PHP присвоит значение вместо ссылки.

См. также

Рецепт 6.3 о передаче значений по ссылке.

6.7. Возвращение более одного значения

Задача

Необходимо вернуть из функции более одного значения.

Решение

Верните массив и используйте функцию `list()` для разделения элементов:

```
function averages($stats) {
    ...
    return array($median, $mean, $mode);
}

list($median, $mean, $mode) = averages($stats);
```

Обсуждение

С точки зрения производительности это не очень хорошая идея. Здесь мы имеем некоторые дополнительные накладные расходы, поскольку PHP должен сначала создать массив, а затем разобрать его. Вот что происходит в этом примере:

```
function time_parts($time) {
    return explode(':', $time);
}

list($hour, $minute, $second) = time_parts('12:34:56');
```

Передается строка времени в том виде, как она выглядит на экране цифровых часов, и вызывается функция `explode()`, чтобы разобрать строку на части в виде элементов массива. К значению, возвращенному функцией `time_parts()`, применяется функция `list()`, чтобы извлечь каждый элемент и занести его в скалярную переменную. Это не очень эффективно, но другие возможные решения еще хуже, т. к. в результате код получится запутанным.

Есть и еще одна возможность – передача значения по ссылке. Однако это несколько неуклюже и делает логически неочевидной передачу необходимых переменных функции. Например:

```
function time_parts($time, &$hour, &$minute, &$second) {
    list($hour, $minute, $second) = explode(':', $time);
}

time_parts('12:34:56', $hour, $minute, $second);
```

Не имея прототипа функции, невозможно, взглянув на этот фрагмент, определить, чем же, по существу, являются переменные `$hour`, `$minute` и `$second`, т. е. значения, возвращаемые функцией `time_parts()`.

Можно также использовать глобальные переменные, но это загромождает глобальное пространство имен и затрудняет возможность определить, какая из переменных была неявно изменена в функции. Например:

```
function time_parts($time) {
    global $hour, $minute, $second;
    list($hour, $minute, $second) = explode(':', $time);
}

time_parts('12:34:56');
```

С другой стороны, в данном случае это очевидно, поскольку определение функции расположено непосредственно перед оператором вызова, но если описание функции находится в другом файле или она написана другим программистом, то функция будет менее понятной и появится возможность появления неуловимых ошибок.

Наш совет состоит в том, что если вы модифицируете значение внутри функции, то возвращайте это значение и присваивайте его переменной, не принимая во внимание другие соображения, такие как значительное увеличение производительности.

См. также

Рецепт 6.3 о передаче значений по ссылке и рецепт 6.11 о видимости переменных.

6.8. Пропуск определенных возвращаемых значений

Задача

Функция возвращает несколько значений, но нам нужны лишь некоторые из них.

Решение

Пропустить переменные позволяет функция `list()`:

```
// Интересуют только минуты
function time_parts($time) {
    return explode(':', $time);
}

list(, $minute,) = time_parts('12:34:56');
```

Обсуждение

Это похоже на ошибку в программе, но фрагмент кода из раздела «Решение» имеет в PHP полное право на существование. Чаще всего это встречается, когда программист выполняет цикл по массиву с помощью функции `each()`, но нужны ему только значения массива:

```
while (list(,$value) = each($array)) {
    process($value);
}
```

Однако оператор `foreach` позволяет написать более понятный код:

```
foreach ($array as $value) {
    process($value);
}
```

В целях уменьшения путаницы мы не очень часто прибегаем к этой функциональности, но если функция возвращает много значений, а нужны только одно или два из них, то этот способ может пригодиться. Вот один из таких примеров – поля считываются с помощью функции `fgetcsv()`, которая возвращает массив, содержащий поля строки. Код выглядит так:

```
while ($fields = fgetcsv($fh, 4096)) {
    print $fields[2] . "\n";  // третье поле
}
```

Если это описанная в коде функция, а не встроенная, то можно определить ключи возвращаемого массива в виде строк, поскольку трудно запомнить, например, что элементу 2 соответствует значение `'rank'`:

```
while ($fields = read_fields($filename)) {
    $rank = $fields['rank']; // третье поле теперь называется rank
    print "$rank\n";
}
```

Однако ниже показан более эффективный метод:

```
while (list(,,$rank,,) = fgetcsv($fh, 4096)) {
    print "$rank\n";          // непосредственно присваиваем $rank
}
```

Будьте внимательны, чтобы не ошибиться при подсчете количества запятых.

См. также

Дополнительную информацию о чтении файла с помощью функции `fgetcsv()` в рецепте 1.9.

6.9. Возвращение информации об ошибке

Задача

Необходимо показать ошибку, произошедшую в результате работы функции.

Решение

Возвращаем значение `false`:

```
function lookup($name) {
    if (empty($name)) { return false; }
    ...
}

if (false !== lookup($name)) { /* реакция на результат поиска */ }
```

Обсуждение

В PHP значения, не относящиеся к истинным, не стандартизованы и легко могут вызвать ошибки. Поэтому лучше всего, если все ваши функции возвращают предопределенное ключевое слово `false`, т. к. оно больше подходит для проверки логического значения.

Другие возможности – это `''` или `0`. Однако, хотя все три значения оцениваются в операторе `if` как неистинные, между ними есть существенная разница. Кроме того, иногда возвращаемое значение `0` представляет значащий результат, а требуется еще возвратить сообщение об ошибке.

Например, функция `strpos()` возвращает позицию в строке первого вхождения подстроки. Если подстрока не найдена, то `strpos()` возвращает значение `false`, а если найдена, – позицию в виде целого числа. Итак, определить расположение подстроки можно следующим образом:

```
if (strpos($string, $substring)) { /* нашли! */ }
```

Однако если `$substring` обнаружена точно в начале строки `$string`, то возвращается значение `0`. К сожалению, внутри оператора `if` это значение оценивается как `false`, поэтому условие не выполняется. Ниже показан корректный способ обработки значения, возвращаемого функцией `strpos()`:

```
if (false !== strpos($string, $substring)) { /* нашли! */ }
```

Кроме того, значение `false` всегда будет ложным – в текущей версии PHP и во всех последующих. Для других значений это не гарантируется. Например, в PHP 3 функция `empty('0')` возвращала значение `true`, но оно было заменено на `false` в PHP 4.

См. также

Введение в главе 5, где представлено более подробное описание истинных значений переменных; документацию по функции `strpos()` на *http://www.php.net/strpos* и по функции `empty()` на *http://www.php.net/empty*; информацию о переходе от PHP 3 к PHP 4 на *http://www.php.net/migration4*.

6.10. Вызов переменных функций

Задача

Необходимо вызывать различные функции в зависимости от значения переменной.

Решение

Используйте переменные переменные:

```
function eat_fruit($fruit) { print "chewing $fruit."; }

$function = 'eat_fruit';
$fruit = 'kiwi';

$function($fruit); // вызов функции eat_fruit()
```

Обсуждение

При наличии нескольких вариантов вызова следует обратиться к ассоциативному массиву имен функций:

```
$dispatch = array(
    'add'      => 'do_add',
    'commit'   => 'do_commit',
    'checkout' => 'do_checkout',
    'update'   => 'do_update'
);

$cmd = (isset($_REQUEST['command']) ? $_REQUEST['command'] : '');

if (array_key_exists($cmd, $dispatch)) {
    $function = $dispatch[$cmd];
    $function(); // вызываем функцию
} else {
    error_log("Unknown command $cmd");
}
```

Вышеприведенный код берет имя команды из запроса и выполняет эту функцию. Обратите внимание на проверку того, что команда входит в перечень допустимых команд. Она предохраняет код от вызова произвольной функции, переданной в запросе, такой как `phpinfo()`. Это делает программу более защищенной и позволяет легко регистрировать ошибки.

Есть и еще одно преимущество – появляется возможность связать различные команды с одной и той же функцией, так что имя может быть и длинным, и коротким:

```
$dispatch = array(
    'add'      => 'do_add',
    'commit'   => 'do_commit',   'ci' => 'do_commit',
    'checkout' => 'do_checkout', 'co' => 'do_checkout',
    'update'   => 'do_update',   'up' => 'do_update'
);
```

См. также

Рецепт 5.4, где подробно описаны переменные переменные.

6.11. Доступ к глобальной переменной внутри функции

Задача

Необходимо получить доступ к глобальной переменной внутри функции.

Решение

Поместите глобальную переменную в локальную область видимости с помощью ключевого слова `global`:

```
function eat_fruit($fruit) {
   global $chew_count;

   for ($i = $chew_count; $i > 0; $i--) {
       ...
   }
}
```

Или сошлитесь на нее непосредственно в массиве `$GLOBALS`:

```
function eat_fruit($fruit) {
   for ($i = $GLOBALS['chew_count']; $i > 0; $i--) {
       ...
   }
}
```

Обсуждение

Если внутри функции используется некоторое количество глобальных переменных, то ключевое слово `global` может сделать синтаксис функции более легким для понимания, особенно если глобальные переменные размещены в строках.

Глобальные переменные можно поместить в локальную область видимости, указав ключевое слово `global` со списком переменных, разделенных запятыми:

```
global $age,$gender,shoe_size;
```

Можно также задавать имена глобальных переменных с помощью переменных переменных:

```
$which_var = 'age';
global $$which_var; // ссылается на глобальную переменную $age
```

Однако если функция `unset()` вызывается для переменной, помещенной в локальную область видимости с помощью ключевого слова `global`, то переменная становится не установленной только внутри функции. Для того чтобы сбросить переменную в глобальной области, надо вызвать функцию `unset()` для элемента массива `$GLOBALS`:

```
$food =  'pizza';
$drink = 'beer';

function party() {
    global $food, $drink;

    unset($food);              // едим пиццу
    unset($GLOBALS['drink']); // пьем пиво
}

print "$food: $drink\n";
party();
print "$food: $drink\n";
pizza: beer
pizza:
```

Видно, что переменная `$food` остается той же самой, в то время как переменная `$drink` стала неустановленной. Объявление переменной глобальной внутри функции подобно присваиванию адреса глобальной переменной локальной переменной:

```
$food = &GLOBALS['food'];
```

См. также

Документацию по областям видимости переменных на *http://www.php.net/variables.scope* и по ссылкам на переменные на *http://www.php.net/language.references*.

6.12. Создание динамических функций

Задача

Необходимо создавать и определять функцию во время выполнения программы.

Решение

Это делается при помощи функции `create_function()`:

```
$add = create_function('$i,$j', 'return $i+$j;');

$add(1, 1); // возвращает 2
```

Обсуждение

Первый параметр функции `create_function()` представляет собой строку, содержащую аргументы функции, а второй параметр – тело функции. Функция `create_function()` работает крайне медленно, поэтому, если возможно заранее определить функцию, лучше так и сделать.

Чаще всего функция `create_function()` используется при разработке пользовательских вариантов функций сортировки `usort()` или `array_walk()`:

```
// сортирует файлы в порядке, обратном обычному
usort($files, create_function('$a, $b', 'return strnatcmp($b, $a);'));
```

См. также

Рецепт 4.17 о функции `usort()`; документацию по функции `create_function()` на *http://www.php.net/create-function* и по функции `usort()` на *http://www.php.net/usort*.

7

Классы и объекты

7.0. Введение

Изначально PHP не был объектно-ориентированным (OO) языком. Но по мере развития объектно-ориентированные возможности стали в нем появляться. Сначала можно было определять классы, но не было конструкторов. Затем появились конструкторы, но не было деструкторов. Медленно, но уверенно, по мере роста количества пользователей, сталкивавшихся с ограничениями синтаксиса PHP, добавлялись новые возможности, призванные удовлетворить требования программистов.

Однако тот, кто хочет, чтобы PHP был настоящим OO языком, возможно, будет разочарован. По своей сути PHP – язык процедурно-ориентированный. Это не Java. Но для тех, кому в программе требуются некоторые OO возможности, вероятно, PHP как раз то, что нужно.

Класс – это пакет, содержащий, во-первых, данные, а во-вторых, методы для доступа или модификации данных. Данные состоят из переменных и известны как *свойства*. Другую часть класса составляют функции, которые могут изменять свойства класса, и называются они *методами*.

Определяя класс, мы не определяем объект, к которому можно получить доступ и которым можно манипулировать. Мы определяем шаблон для объекта. По этому шаблону создаются послушные объекты в процессе так называемого создания *экземпляра класса*. Программа может иметь несколько объектов одного и того же класса, точно так же, как у человека может быть более одной книги или не один фрукт, а несколько.

Классы также организованы в определенную иерархию. Вверху цепочки находится родовой класс. В PHP этому классу дано имя `stdClass`, «стандартный» класс. Каждый нижележащий класс более специализирован, чем его родитель. Например, родительский класс может быть зданием. А здания уже могут подразделяться на жилые и промышленные. Жилые здания можно разделить на индивидуальные и многоквартирные и т. д.

И те и другие имеют тот же самый набор свойств, что и жилые здания вообще, точно так же, как у жилых и коммерческих строений есть что-то общее. О родственных отношениях классов говорят, что дочерний класс наследует свойства и методы родительского. Это делает возможным повторное использование кода родительского класса, и надо лишь написать код, адаптирующий новый дочерний класс к его специализации. Это называется наследованием и является одним из главных преимуществ классов перед функциями. Процесс определения дочернего класса известен как создание *подкласса*, или *расширение*.

Объекты в PHP, помимо своей обычной ОО роли, играют и другую. PHP не может использовать более одного пространства имен, поэтому способность класса упаковывать несколько свойств в один объект исключительно полезна. Она позволяет четко разграничивать различные области переменных.

Объявить и создать классы в PHP нетрудно:

```
class guest_book {
    var $comments;
    var $last_visitor;

    function update($comment, $visitor) {
        ...
    }

}
```

Ключевое слово `class` определяет класс точно так же, как слово `function` определяет функцию. Свойства объявляются с помощью ключевого слова `var`. Объявление метода идентично объявлению функции.

Ключевое слово `new` создает экземпляр объекта:

```
$gb = new guest_book;
```

Более подробно реализация объекта описывается в рецепте 7.1.

Внутри класса можно объявить свойства с помощью ключевого слова `var`. Делать это не обязательно, но полезно к этому привыкнуть. Поскольку PHP не требует предварительного объявления всех переменных, то можно создать их внутри класса, и при этом PHP не выдаст ошибки или какого-либо предупреждения. Поэтому список переменных в начале определения класса может ввести в заблуждение, так как он может не совпадать с реальным списком переменных класса.

Помимо объявления свойства, можно также присвоить ему значение:

```
var $last_visitor = 'Donnan';
```

Следующая конструкция позволяет присвоить только постоянное значение:

```
var $last_visitor = 'Donnan';          // правильно
var $last_visitor = 9;                 // правильно
var $last_visitor = array('Jesse');    // правильно
```

```
var $last_visitor = pick_visitor();   // неправильно
var $last_visitor = 'Chris' . '9';    // неправильно
```

Если вы попытаетесь присвоить что-нибудь еще, то PHP прекратит работу с синтаксической ошибкой.

Для присваивания непостоянного значения применяется внутренний метод класса.

```
var $last_visitor;

function update($comment, $visitor) {
    if (!empty($comment)) {
        array_unshift($this->comments, $comment);
        $this->last_visitor = $visitor;
    }
}
```

Если посетитель оставил свой комментарий, то вы добавляете его в начало массива комментариев и отмечаете этого посетителя в гостевой книге как последнего. Переменная `$this` – это специальная переменная, ссылающаяся на текущий объект. Поэтому, чтобы получить доступ к свойству `$size` объекта изнутри этого объекта, сошлитесь на `$this->size`.

Для того чтобы переменная получала значение во время создания экземпляра объекта, его надо присваивать в конструкторе класса. Конструктор *класса* – это метод, автоматически вызываемый при создании объекта и имеющий то же самое имя, что и класс:

```
class guest_book {
    var $comments;
    var $last_visitor;

    function guest_book($user) {
        $dbh  =  mysql_connect('localhost', 'username', 'password');
        $db   =  mysql_select_db('sites');
        $user =  mysql_real_escape_string($user);
        $sql  = "SELECT comments, last_visitor FROM
                 guest_books WHERE user='$user'";
        $r    =  mysql_query($sql);

        if ($obj  = mysql_fetch_object($r)) {
            $this->comments = $obj->comments;
            $this->last_visitor = $obj->last_visitor;
        }
    }
}

$gb = new guest_book('stewart');
```

Конструкторам посвящен рецепт 7.2. Учтите, что функция `mysql_real_escape_string()` впервые появилась в версии PHP 4.3; в более ранних версиях следует вызывать функцию `mysql_escape_string()`.

Будьте внимательны, чтобы по ошибке не напечатать `$this->$size`. Это допустимо, но не то же самое, что `$this->size`. Иначе получится свойство, имя которого представляет значение переменной `$size`. Скорее всего, переменная `$size` не определена, поэтому `$this->$size` покажет пустое значение. За более подробной информацией о переменных именах свойств обращайтесь к рецепту 6.5.

Для доступа к методу или члену переменной помимо символа `->` можно использовать и символ `::`. Этот синтаксис позволяет получить доступ к статическим методам класса. Эти методы одинаковы для каждого экземпляра класса, поскольку они не могут зависеть от данных, характерных только для конкретного экземпляра. Например:

```
class convert {
    // преобразование из градусов по Цельсию в градусы по Фаренгейту
    function c2f($degrees) {
        return (1.8 * $degrees) + 32;
    }
}

$f = convert::c2f(100); // 212
```

Чтобы реализовать наследование путем расширения существующего класса, применяется ключевое слово `extends`:

```
class xhtml extends xml {

}
```

Дочерний класс наследует методы родителя и может произвольно создавать свои собственные версии этих методов:

```
class DB {
    var $result;

    function getResult() {
        return $this->result;
    }

    function query($sql) {
        error_log("query() must be overridden by a database-specific child");
        return false;
    }
}

class MySQL extends DB {
    function query($sql) {
        $this->result =  mysql_query($sql);
    }
}
```

Приведенный выше класс `MySQL` наследует неизмененный метод `getResult()` родительского класса DB, но имеет свою собственную, специфичную для MySQL версию метода `query()`.

Предваряйте имя метода строкой `parent::` для явного вызова родительского метода:

```
function escape($sql) {
    $safe_sql = mysql_real_escape_string($sql); // переводим специальные
                                                // символы в escape-последовательности
    $safe_sql = parent::escape($safe_sql);      // родительский метод
                                                // добавляет `` вокруг $sql
    return $safe_sql;
}
```

Рецепт 7.7 посвящен доступу к переопределенным методам.

Лежащий в основе мощи PHP механизм называется Zend. PHP 4 использует Zend Engine 1; PHP 5 будет основан на усовершенствованной версии – Zend Engine 2 (ZE2). ZE2 имеет совершенно новую объектную модель, которая поддерживает массу новых объектно-ориентированных возможностей: конструкторы и деструкторы, частные методы, обработку исключений и вложенные классы. В этой главе мы отмечаем, где есть отличия в синтаксисе или функциональности между PHP 4 и тем, что поддерживается в ZE2, поэтому вы сможете планировать свои действия на будущее.

7.1. Реализация объектов

Задача

Необходимо создать новый экземпляр класса.

Решение

Определите класс, затем укажите ключевое слово `new` для создания нового экземпляра класса:

```
class user {
    function load_info($username) {
       // загружаем учетную запись из базы данных
    }
}

$user = new user;
$user->load_info($_REQUEST['username']);
```

Обсуждение

Можно создать несколько экземпляров одного и того же класса:

```
$adam = new user;
$adam->load_info('adam');

$dave = new user;
$dave->load_info('adam');
```

Здесь два независимых объекта, которые случайно могут содержать идентичную информацию. Они подобны однояйцевым близнецам; могут появиться в одно время, но проживают различные жизни.

См. также

Подробную информацию о копировании объектов в рецепте 7.4; рецепт 7.5, где подробно описано копирование объектов по ссылке; документацию по классам и объектам на *http://www.php.net/oop*.

7.2. Определение конструкторов объектов

Задача

Необходимо определить метод, вызываемый во время создания нового экземпляра класса. Например, требуется автоматически загружать информацию из базы данных во время создания объекта.

Решение

Определите метод с тем же самым именем, что и имя класса:

```
class user {
    function user($username, $password) {
        ...
    }
}
```

Обсуждение

Если функция имеет то же имя, что и класс, она действует как конструктор:

```
class user {
    var $username;

    function user($username, $password) {
        if ($this->validate_user($username, $password)) {
            $this->username = $username;
        }
    }
}

$user = new user('Grif', 'Mistoffelees'); // используем встроенный конструтор
```

PHP не всегда поддерживал конструкторы. Поэтому программисты создавали псевдоконструкторы, следуя соглашению об именах и вызывая эти функции после создания объекта:

```
class user {
    ...

    init($username, $password) { ... }
```

```
}

$user = new user();
$user->init($username, $password);
```

Увидев что-нибудь подобное, знайте, что это, скорее всего, унаследованный код.

Однако стандартное наименование всех конструкторов облегчает вызов родительских конструкторов (поскольку нет необходимости знать имя родительского класса), а также не требует модификации конструктора, если изменяется имя класса. В Zend Engine 2 соглашения об именах конструкторов были изменены, и новое имя конструктора теперь `__construct()`. Однако в целях обратной совместимости, если такой метод не найден, PHP пытается вызвать конструктор с тем же именем, что и имя класса.

См. также

Рецепт 7.7, в котором подробно рассказывается о вызове родительских конструкторов; документацию по конструкторам объектов на *http://www.php.net/oop.constructor*.

7.3. Уничтожение объекта

Задача

Необходимо удалить объект.

Решение

Объекты автоматически уничтожаются, когда сценарий заканчивает работу. Для немедленного уничтожения объекта предназначена функция `unset()`:

```
$car = new car; // покупаем новую машину
...
unset($car);    // машина попала в аварию
```

Обсуждение

Как правило, нет необходимости удалять объекты вручную, но в случае больших циклов функция `unset()` иногда помогает удержать контроль над расходованием памяти.

В PHP 4 нет деструкторов, однако Zend Engine 2 поддерживает их с помощью метода `__destruct()`.

См. также

Документацию по функции `unset()` на *http://www.php.net/unset*.

7.4. Клонирование объектов

Задача

Необходимо создать копию существующего объекта. Например, есть объект, содержащий сообщение для рассылки, и вы хотите скопировать его в качестве основы для ответного сообщения.

Решение

Для присваивания объекта новой переменной применяется оператор =:

```
$rabbit = new rabbit;
$rabbit->eat();
$rabbit->hop();
$baby = $rabbit;
```

Обсуждение

В PHP для создания копии объекта достаточно присвоить его новой переменной. Начиная с этого момента каждый экземпляр объекта живет независимой жизнью, и изменение одного из них не оказывает влияния на другой:

```
class person {
    var $name;

    function person ($name) {
        $this->name = $name;
    }
}

$adam = new person('adam');
print $adam->name;    // adam
$dave = $adam;
$dave->name = 'dave';
print $dave->name;    // dave
print $adam->name;    // все еще adam
```

Zend Engine 2 допускает явное клонирование объекта с помощью метода `__clone()`, вызываемого при каждом копировании объекта. Это обеспечивает более тонкое управление набором копируемых свойств.

См. также

Подробную информацию о присваивании объектов по ссылке в рецепте 7.5.

7.5. Присваивание ссылок на объекты

Задача

Необходимо связать два объекта, так чтобы при обновлении одного из них, аналогично изменялся бы и другой.

Решение

Для присваивания одного объекта другому по ссылке применяется оператор =&:

```
$adam = new user;
$dave =& $adam;
```

Обсуждение

В результате присваивания объекта с помощью оператора = создается новая копия объекта. Поэтому изменение одного не влияет на другой. Но в случае применения оператора =& два объекта указывают друг на друга, поэтому любые изменения одного объекта отражаются на втором:

```
$adam = new user;
$adam->load_info('adam');

$dave =& $adam;
$dave->load_info('dave');
```

Значения в объекте $adam равны значениям в объекте $dave.

См. также

Рецепт 7.4, где более подробно описывается копирование объектов; документацию по ссылкам на *http://www.php.net/references*.

7.6. Применение методов к объекту, возвращенному другим методом

Задача

Необходимо вызвать метод для объекта, возвращенного другим методом.

Решение

Присвойте объект временной переменной, а затем вызовите метод для этой временной переменной:

```
$orange = $fruit->get('citrus');
$orange->peel();
```

Обсуждение

Это необходимо, поскольку следующее выражение приведет к синтаксической ошибке:

```
$fruit->get('citrus')->peel();
```

Zend Engine 2 поддерживает непосредственную разадресацию объектов, возвращенных методом, поэтому в таком обходном маневре больше нет необходимости.

7.7. Доступ к переопределенным методам

Задача

Необходимо получить доступ к методам в родительском классе, которые были переопределены в дочернем классе.

Решение

Добавьте префикс `parent::` к имени метода:

```
class shape {
    function draw() {
        // выводим на экран
    }
}

class circle extends shape {
   function draw($origin, $radius) {
      // проверка данных
      if ($radius > 0) {
          parent::draw();
          return true;
      }

      return false;
   }
}
```

Обсуждение

Если подменить родительский метод путем определения его в дочернем методе, то родительский метод не будет вызван до тех пор, пока вы не сошлетесь на него явно.

В рассмотренном решении мы переопределяем метод `draw()` в дочернем классе `circle`, поскольку необходимо принять специфические для окружности параметры и проверить данные. Однако в данном случае мы еще хотим вызвать базовую функцию `shape::draw()`, которая фактически выполняет рисование, поэтому вызываем функцию `parent::draw()` внутри метода, если значение переменной `$radius` больше 0.

Префикс `parent::` может применяться только внутри класса. Вызов функции `parent::draw()` вне класса приведет к синтаксической ошибке. Например, если функция `circle::draw()` проверяла только радиус, а необходимо было вызвать также и функцию `shape::draw()`, то ничего бы не получилось:[1]

```
$circle = new circle;
if ($circle->draw($origin, $radius)) {
```

[1] На самом деле это приводит к сбою с ошибкой `unexpected T_PAAMAYIM_NEKUDOTAYIM`, что на иврите означает «двойное двоеточие».

```
    $circle->parent::draw();
}
```

Если необходимо вызвать конструктор, принадлежащий к родительскому объекту, но неизвестно имя родительского класса, вызовите функцию `get_parent_class()` для динамического определения имени родителя, а затем объедините его с префиксом `parent::`, чтобы вызвать родительский конструктор:

```
class circle extends shape {

    function circle() {
        $parent = get_parent_class($this);
        parent::$parent();
    }
}
```

Функция `get_parent_class()` принимает имя класса или объект, а возвращает имя родителя объекта. Для поддержки общности передавайте `$this`, что является ссылкой на текущий объект. В этом случае возвращает `shape`. Затем используйте префикс `parent::`, чтобы быть уверенным в том, что PHP явно вызывает конструктор родительского класса. Вызывая `$parent()` без префикса `parent::`, вы рискуете вызвать метод класса `circle`, который подменяет родительское определение.

Вызов функции `parent::$parent()` может выглядеть несколько странно. Однако PHP всего лишь заменяет переменную `$parent` на имя родительского класса. Теперь, поскольку после переменной стоят круглые скобки, PHP знает, что он должен вызвать метод.

Можно жестко прописать вызов функции `parent::shape()` прямо в конструкторе класса `circle`:

```
function circle() {
    parent::shape();
}
```

Однако это не так гибко, как вызов функции `get_parent_class()`, хотя и быстрее, поэтому если известно, что иерархия объектов не будет изменяться, то такая замена может дать преимущество.

В заключение скажем, что нельзя составлять цепочки ключевых слов `parent::`, чтобы вернуться к классу прародителя, поэтому `parent::parent::foo()` не работает.

См. также

Дополнительные сведения о конструкторах объектов в рецепте 7.2; документацию по родительским классам на *http://www.php.net/keyword.parent* и о функции `get_parent_class()` на *http://www.php.net/get-parent-class*.

7.8. Перегрузка свойств

Задача

Необходимо, чтобы функции обработчики выполнялись при каждом чтении или записи свойств объекта. Это позволяет написать универсальный код для управления доступом к свойствам класса.

Решение

Это делается при помощи экспериментального расширения перегрузки; кроме того, надо написать методы `__get()` и `__set()` для перехвата запросов свойств.

Обсуждение

Перегрузка свойств позволяет без труда скрыть от пользователя истинное расположение свойств объекта и структуру данных, в которой они хранятся.

Например, класс `pc_user`, приведенный в примере 7.1, хранит переменные в массиве `$data`.

Пример 7.1. pc_user class

```
require_once 'DB.php';

class pc_user {

    var $data = array();

    function pc_user($user) {
        /* соединяемся с базой данных и загружаем информацию
         * о пользователе по имени $user в $this->data
         */

         $dsn = 'mysql://user:password@localhost/test';
         $dbh = DB::connect($dsn);
         if (DB::isError($dbh)) { die ($dbh->getMessage()); }

         $user = $dbh->quote($user);
         $sql = "SELECT name,email,age,gender FROM
                 users WHERE user LIKE '$user'";
         if ($data = $dbh->getAssoc($sql)) {
             foreach($data as $key => $value) {
                 $this->data[$key] = $value;
             }
         }
    }

    function __get($property_name, &$property_value) {
        if (isset($this->data[$property_name])) {
            $property_value = $this->data[$property_name];
            return true;
```

```
        }

        return false;
    }

    function __set($property_name, $property_value) {
        $this->data[$property_name] = $property_value;
        return true;
    }
}
```

Ниже показано, как использовать класс `pc_user`:

```
overload('pc_user');

$user = new pc_user('johnwood');
$name = $user->name;                // читает $user->data['name']
$user->email = 'jonathan@wopr.mil'; // устанавливает  $user->data['email']
```

Конструктор класса подсоединяется к таблице `users` в базе данных и извлекает информацию о пользователе по имени `$user`. При установке данных функция `__set()` переписывает элементы массива `$data`. Аналогично применяется функция `__get()`, чтобы перехватить вызов и возвратить правильный элемент массива.

Использование массива в качестве альтернативного исходного хранилища переменных не дает большого преимущества перед объектом без перегрузки, но эта функциональность не ограничена простыми массивами. Например, можно заставить `$this->email` вернуть метод `get_name()` объекта email. Можно также избежать извлечения всей информации о пользователе за один раз и запрашивать ее по мере необходимости. Другой альтернативой является использование более устойчивого механизма хранения данных, такого как файлы, разделяемая память или база данных.

См. также

Рецепт 6.7, содержащий информацию о хранении объектов во внешних источниках; документацию по расширению перегрузки на *http://www.php.net/overload*.

7.9. Полиморфизм методов

Задача

Необходимо передать управление тому или иному коду в зависимости от количества и типа аргументов, переданных методу.

Решение

PHP не поддерживает полиморфизм методов в качестве встроенной функциональности. Однако можно его эмулировать посредством раз-

личных функций проверки типа. Следующая функция, `combine()`, использует функции `is_numeric()`, `is_string()`, `is_array()` и `is_bool()`:

```
// функция combine() складывает числа, соединяет строки, объединяет массивы
// и выполняет операцию AND над битовыми и логическими аргументами
function combine($a, $b) {
    if (is_numeric($a) && is_numeric($b)) {
        return $a + $b;
    }

    if (is_string($a)  && is_string($b))  {
        return "$a$b";
    }

    if (is_array($a)   && is_array($b))   {
        return array_merge($a, $b);
    }

    if (is_bool($a)    && is_bool($b))    {
        return $a & $b;
    }

    return false;
}
```

Обсуждение

PHP не позволяет объявлять тип переменной в прототипе метода, поэтому он не может реализовать условное выполнение различных методов на основе их сигнатур, как это делается в Java и C++. Вместо этого можно написать одну функцию и использовать оператор `switch`, чтобы вручную восстановить эту функциональность.

Например, PHP позволяет редактировать образы с помощью GD. Удобно, если класс образа способен передать или местоположение образа (удаленное или локальное) или дескриптор, который PHP назначает существующему потоку образа. В примере 7.2 показан класс `pc_Image`, который именно это и делает.

Пример 7.2. pc_Image class

```
class pc_Image {

    var $handle;

    function ImageCreate($image) {
        if (is_string($image)) {
            // простое определение типа

            // путем захвата суффикса файла
            $info = pathinfo($image);
            $extension = strtolower($info['extension']);
            switch ($extension) {
            case 'jpg':
            case 'jpeg':
```

```
                $this->handle = ImageCreateFromJPEG($image);
                break;
            case 'png':
                $this->handle = ImageCreateFromPNG($image);
                break;
            default:
                die('Images must be JPEGs or PNGs.');
            }
        } elseif (is_resource($image)) {
            $this->handle = $image;
        } else {
            die('Variables must be strings or resources.');
        }
    }
}
```

В данном случае любая переданная строка трактуется как путь к файлу, поэтому мы используем функцию `pathinfo()` для перехвата расширения файла. Как только расширение становится известным, мы пытаемся определить, какая из функций `ImageCreateFrom()` безошибочно открывает образ и создает дескриптор.

Если это не строка, то мы обращаемся непосредственно к потоку, который имеет тип `resource`. Поскольку нет необходимости проводить преобразование, мы присваиваем поток непосредственно переменной `$handle`. Естественно, применение этого класса при разработке программного обеспечения позволяет добиться более надежного определения и обработки ошибок.

Полиморфизм методов также позволяет вызывать методы с различным количеством аргументов. Код, который определяет количество аргументов внутри метода, идентичен тому, как происходит обработка функций с переменным количеством аргументов с помощью `func_num_args()`. Это рассматривается в рецепте 6.5.

См. также

Рецепт 6.5 о функциях с переменным количеством аргументов; документация по функции `is_string()` на *http://www.php.net/is-string*, по функции `is_resource()` на *http://www.php.net/is-resource* и по функции `pathinfo()` на *http://www.php.net/pathinfo*.

7.10. Обнаружение методов и свойств объекта

Задача

Необходимо просмотреть объект, чтобы определить, какие методы и свойства у него есть, что позволяет написать код, работающий с любыми родовыми объектами независимо от типа.

Решение

Для исследования объекта и получения информации о нем применяются функции `get_class_methods()` и `get_class_vars()`:

```
// изучаем машины
$car_methods = get_class_methods('car');
$car_vars    = get_class_vars('car');

// действуем на основании полученных знаний
if (in_array('speed_away', $car_methods)) {
    $getaway_van = new car;
    $getaway_van->speed_away();
}
```

Обсуждение

Редко бывает, когда есть объект, но невозможно исследовать его фактический исходный код, чтобы разобраться в его работе. Все же эти функции могут оказаться полезными для проектов, в которых необходимо применить целый ряд различных классов, таких как автоматическое создание документации по классам, отладчиков родовых объектов и хранителей состояний, подобных функции `serialize()`.

И функция `get_class_methods()`, и функция `get_class_vars()` возвращают массив значений. В функции `get_class_methods()` ключи – это числа, а значения – это имена методов. В случае функции `get_class_vars()` возвращаются и имена переменных, и значения по умолчанию (присвоенные с помощью конструкции `var`), при этом имена переменных возвращаются как ключи, а значения по умолчанию, если таковые есть, – как значения.

Другая полезная функция – это `get_object_vars()`. В отличие от своей сестры, функции `get_class_vars()`, функция `get_object_vars()` возвращает информацию о переменных конкретного экземпляра объекта, а не предка вновь созданного объекта.

Поэтому с ее помощью можно проверить статус объекта, то есть его текущее состояние в программе:

```
$clunker = new car;
$clunker_vars = get_object_vars($clunker); // мы передаем объект, а не класс
```

Нам нужна информация о конкретном объекте, поэтому передается объект, а не имя его класса. Но функция `get_object_vars()` возвращает информацию в том же формате, что и функция `get_class_vars()`.

Это позволяет без труда написать короткий сценарий, чтобы посмотреть, как добавляются переменные класса:

```
$new_vars = array_diff(array_keys(get_object_vars($clunker)),
                       array_keys(get_class_vars('car')));
```

Сначала при помощи функции array_keys() извлекаем имена переменных. Затем, вызвав функцию array_diff(), определяем, какие из переменных объекта $clunker не определены в классе car.

Если вам достаточно лишь взглянуть на экземпляр объекта, и вы не хотите разбираться с функцией get_class_vars(), то для печати значений объекта обратитесь либо к функциям var_dump(), var_export(), либо к функции print_r(). Каждая из них выводит информацию на печать немного по-разному; функция var_export(), по вашему выбору, может возвращать информацию, не отображая ее.

См. также

Более подробную информацию о выводе переменных в рецепте 5.8; документацию по функции get_class_vars() на *http://www.php.net/get-class-vars*, по функции get_class_methods() на *http://www.php.net/get-class-methods*, по функции get_object_vars() на *http://www.php.net/get-object-vars*, по функции var_dump() на *http://www.php.net/var-dump*, по функции var_export() на *http://www.php.net/var-export* и по функции print_r() на *http://www.php.net/print-r*.

7.11. Добавление свойств в базовый объект

Задача

Необходимо создать объект и добавить в него свойства, но не определяя его формально как отдельный класс. Это удобно, когда нужна функция, требующая объект с определенными свойствами, например такой, который возвращает функция mysql_fetch_object() или функция imap_header().

Решение

Это делается при помощи встроенного базового класса stdClass:

```
$pickle = new stdClass;
$pickle->type = 'fullsour';
```

Обсуждение

Точно так же, как функция array() возвращает пустой массив, создание объекта типа stdClass предоставляет объект без свойств и методов.

Как и в случае объектов, принадлежащих другим классам, можно создавать новые свойства объекта, присваивать им значения и проверять эти свойства:

```
$guss = new stdClass;

$guss->location = 'Essex';
print "$guss->location\n";
```

```
$guss->location = 'Orchard';
print "$guss->location\n";
Essex
Orchard
```

Однако после того как создан экземпляр объекта, методы добавлять нельзя.

Создание объекта типа stdClass полезно, когда нужна функция, принимающая базовый объект, такой, который возвращает функция, делающая выборку из базы данных, но вы не хотите посылать запрос в базу данных. Например:

```
function pc_format_address($obj) {
    return "$obj->name <$obj->email>";
}

$sql = "SELECT name, email FROM users WHERE id=$id";
$dbh = mysql_query($sql);
$obj = mysql_fetch_object($dbh);
print pc_format_address($obj);
David Sklar <david@example.com>
```

Функция pc_print_address() принимает имя и адрес электронной почты и преобразует эти значения в формат, необходимый для полей To и From в почтовой программе. Ниже показано, как вызывать такую функцию, не вызывая функцию mysql_fetch_object():

```
$obj = new stdClass;
$obj->name = 'Adam Trachtenberg';
$obj->email = 'adam@example.com';
print pc_format_address($obj);
Adam Trachtenberg <adam@example.com>
```

7.12. Динамическое создание класса

Задача

Необходимо создать класс, о котором не все известно до момента выполнения программы.

Решение

Это делается при помощи функции eval() со встроенными переменными:

```
eval("class van extends $parent_class {
    function van() {
        \$this->$parent_class();
    }
};");

$mystery_machine = new van;
```

Обсуждение

В то время как использование в PHP имен переменных для вызова функций или создания объектов вполне допустимо, определять таким же образом функции и классы нельзя:

```
$van();                     // правильно
$van = new $parent_class    // правильно
function $van() {};         // не правильно
class $parent_class {};     // не правильно
```

Попытка выполнения любого из последних двух примеров приведет к синтаксической ошибке, поскольку PHP ожидает строку, а ему предлагают переменную.

Поэтому если необходимо создать класс с именем `$van`, но заранее неизвестно, что будет храниться в `$van`, следует применить функцию `eval()`, которая проделает всю «грязную» работу:

```
eval("class $van {};");
```

Каждый вызов функции `eval()` сильно снижает производительность, поэтому для сайтов с большим трафиком надо постараться так реструктурировать код, чтобы по возможности избежать таких приемов. Кроме того, если определение класса основано на вводе пользователей, следует предотвратить употребление любых потенциально опасных символов.

См. также

Рецепт 7.13 о динамической реализации объектов; документацию по функции `eval()` на *http://www.php.net/eval*.

7.13. Динамическая реализация объекта

Задача

Необходимо создать экземпляр объекта, но до выполнения программы имя его класса неизвестно. Например, требуется выполнить локализацию сайта с помощью создания объекта, относящегося к определенному языку. Однако пока страница не запрошена, неизвестно, какой язык выбрать.

Решение

Используйте переменную в качестве имени класса:

```
$language = $_REQUEST['language'];
$valid_langs = array('en_US' => 'US English',
                     'en_GB' => 'British English',
                     'es_US' => 'US Spanish',
```

```
                        'fr_CA' => 'Canadian French');

if (isset($valid_langs[$language]) && class_exists($language)) {
    $lang = new $language;
}
```

Обсуждение

Иногда имя класса, который требуется реализовать во время выполнения программы, неизвестно, но может быть известна часть его имени. Например, для создания иерархии псевдоимен классов перед именами классов можно помещать префикс из последовательности символов. Вот почему мы часто используем строку `pc_` для представления книги «PHP. Сборник рецептов» (PHP Cookbook), а PEAR использует `Net_` перед именами всех сетевых классов.

Однако в то время как следующий фрагмент допустим в PHP:

```
$class_name = 'Net_Ping';
$class = new $class_name;               // новый Net_Ping
```

то этот — нет:

```
$partial_class_name = 'Ping';
$class = new "Net_$partial_class_name"; // новый Net_Ping
```

Тем не менее следующее верно:

```
$partial_class_name = 'Ping';
$class_prefix = 'Net_';

$class_name = "$class_prefix$partial_class_name";
$class = new $class_name;               // новый Net_Ping
```

Поэтому нельзя создать экземпляр объекта, если имя его класса определено путем конкатенации переменных на том же шаге. Однако поскольку разрешается использовать простые имена переменных, то решить проблему помогает предварительное составление имени класса.

См. также

Рецепт 6.4, где подробно описаны переменные; подробную информацию о динамическом определении вызова в рецепте 7.12; документацию по функции `class_exists()` на *http://www.php.net/class-exists*.

8

Основы Web

8.0. Введение

Возможно, именно веб-программирование является причиной вашего интереса к этой книге. Потребность в специальном языке веб-программирования и послужила причиной написания первой версии PHP и обусловливает растущую популярность этого языка сегодня. На PHP легко писать динамические веб-программы, которые могут выполнять практически любые необходимые операции. Другие главы этой книги посвящены различным возможностям PHP, таким как графика, регулярные выражения, доступ к базам данных и файловый ввод/вывод. Все эти возможности – часть веб-программирования, но предметом этой главы являются специфические для Сети концепции. Рассмотренные здесь темы сделают ваше веб-программирование более строгим и надежным.

В рецептах 8.1, 8.2 и 8.3 показано, как устанавливать, читать и удалять cookies. Cookie – это небольшая текстовая строка, которую по приказу сервера броузер посылает вместе со своими запросами. Обычно HTTP-запросы не могут сохранять свое состояние; каждый запрос нельзя связать с предыдущим. Однако cookie может связать воедино различные запросы одного и того же пользователя. Это упрощает предоставление таких возможностей, как корзина покупателя или отслеживание истории поисковых запросов пользователя.

В рецепте 8.4 показано, как перенаправить пользователя не на ту веб-страницу, которую он запросил, а на другую. В рецепте 8.5 разъясняется работа с модулем сеанса, позволяющим без труда сохранять данные для каждого посетителя, пока он путешествует по сайту. Рецепт 8.6 демонстрирует, как сохранять информацию о сеансе в базе данных, что позволяет улучшить масштабируемость и увеличить гибкость веб-сайта. Определение возможностей пользовательского броузера представлено в рецепте 8.7. Рецепт 8.8 показывает детали конструирования URL, включающего в себя строку запроса GET, а также правильное

кодирование специальных символов и обработку HTML-примитивов (entities).

Следующие два рецепта демонстрируют, как использовать аутентификацию, которая позволяет осуществить парольную защиту веб-страниц. Специальные возможности PHP по работе с базовой HTTP-аутентификацией объясняются в рецепте 8.9. Иногда лучше применять собственные методы аутентификации с применением cookies, как показано в рецепте 8.10.

Три следующих рецепта посвящены управлению выводом. Рецепт 8.11 показывает, как осуществить немедленную посылку выходной информации броузеру. Рецепт 8.12 объясняет функции буферизации вывода. Буфер вывода позволяет перехватить выходную информацию, которая, в противном случае, была бы отображена, или задержать вывод до окончания полной обработки страницы. Автоматическое сжатие выходной информации показано в рецепте 8.13.

Рецепты с 8.14 по 8.19 посвящены обработке ошибок, в том числе управлению потоками вывода, написанию пользовательских функций обработки ошибок и добавлению в программы возможности вывода вспомогательных отладочных сообщений. Рецепт 8.18 включает методы, позволяющие избежать обычного сообщения об ошибке «headers already sent» (заголовки уже посланы), такие как использование буфера вывода, обсуждаемого в рецепте 8.12.

Следующие четыре рецепта показывают, как взаимодействовать с внешними переменными: переменными окружения и конфигурационными установками PHP. В рецептах 8.20 и 8.21 обсуждаются переменные окружения, а рецепты 8.22 и 8.23 посвящены чтению и изменению конфигурационных установок PHP. Если ваш веб-сервер – Apache, то взаимодействие с другими модулями Apache из вашей PHP-программы можно организовать с помощью приемов, описанных в рецепте 8.24.

В рецепте 8.25 показаны некоторые методы профилирования кода и расстановки контрольных точек. Обнаружение участка программы, выполнение которого требует наибольшего времени, позволяет сосредоточить усилия разработчика на улучшении кода, сильнее всего влияющего на скорость обслуживания пользователей.

Эта глава также содержит две программы, помогающие поддерживать веб-сайт. Программа из рецепта 8.26 подтверждает достоверность учетных записей пользователей путем посылки каждому новому пользователю сообщения по электронной почте со специальной ссылкой. Если пользователь не посетит указанную ссылку в течение недели после получения сообщения, то его учетная запись будет удалена. Программа из рецепта 8.27 контролирует запросы каждого пользователя в реальном времени и блокирует запросы пользователей, создающих повышенный трафик на сайте.

8.1. Установка cookies

Задача

Необходимо установить cookie.

Решение

Это делается с помощью функции `setcookie()`:

```
setcookie('flavor','chocolate chip');
```

Обсуждение

Cookies посылаются вместе с HTTP-заголовками, поэтому функцию `setcookie()` необходимо вызывать до того, как сформирована выходная информация.

Для управления поведением cookies можно передать в функцию `setcookie()` дополнительные аргументы. Третий аргумент функции `setcookie()` – это время истечения срока действия в формате метки времени UNIX. Например, время действия следующего cookie истекает в полдень GMT 3 декабря 2004 года:

```
setcookie('flavor','chocolate chip',1102075200);
```

Если третий аргумент функции `setcookie()` опущен (или пустой), то время действия cookie заканчивается в момент закрытия броузера. Следует иметь в виду, что многие системы не могут обрабатывать время истечения срока действия, превышающее 2 147 483 647, поскольку это наибольшее значение метки времени UNIX, которую может содержать 32-битное целое число, как это обсуждалось во введении в главу 3.

Четвертым аргументом функции `setcookie()` является путь. Cookie посылается обратно серверу только в том случае, если путь к запрашиваемым страницам начинается с указанной в аргументе строки. Так, следующий cookie будет послан обратно на страницу, путь к которой начинается со строки */products/*:

```
setcookie('flavor','chocolate chip','','/products/');
```

Страница, которая устанавливает этот cookie, не обязана иметь URL, начинающийся со строки */products/*, но следующий cookie будет послан обратно только на страницу, удовлетворяющую этому требованию.

Пятый аргумент функции `setcookie()` – это домен. Cookie посылается обратно серверу только в том случае, если имя хоста, с которого запрашиваются страницы, заканчивается строкой с именем указанного домена. В следующем фрагменте кода первый cookie посылается обратно всем хостам, принадлежащим домену *example.com*, но второй cookie посылается только в запросах к хосту *jeannie.example.com*:

```
setcookie('flavor','chocolate chip','','','.example.com');
setcookie('flavor','chocolate chip','','','jeannie.example.com');
```

Если бы домен первого cookie был только *example.com*, а не *.example.com*, то он был бы послан только одному хосту *example.com* (а не *www.example.com* или *jeannie.example.com*).

Последний необязательный аргумент `setcookie()` – это установленный в 1 флаг, который приказывает посылать cookie только посредством SSL-соединения. Это может оказаться полезным, если cookie содержит важную информацию, но помните, что на компьютере пользователя информация cookie хранится в открытом виде.

Различные броузеры обрабатывают cookies немного по-разному, особенно в зависимости от того, как строго они подходят к строкам пути и домена и как распределяют приоритеты между cookies с одинаковыми именами. Эти отличия хорошо разъясняются на странице помощи по функции `setcookie()` оперативного руководства.

См. также

Рецепт 8.2, где показано, как читать значения cookie; рецепт 8.3, посвященный удалению cookies; рецепт 8.12, который демонстрирует буферизацию выходной информации; рецепт 8.18, объясняющий, как избежать сообщения об ошибке «headers already sent», которое иногда появляется при вызове функции `setcookie()`; документацию по функции `setcookie()` на *http://www.php.net/setcookie*; расширенную спецификацию cookie, подробно изложенную в RFC 2965 на *http://www.faqs.org/rfcs/rfc2965.html*.

8.2. Чтение значений cookie

Задача

Необходимо прочитать ранее установленное значение cookie.

Решение

Загляните в суперглобальный массив `$_COOKIE`:

```
if (isset($_COOKIE['flavor'])) {
    print "You ate a ".$_COOKIE['flavor']." cookie.";
}
```

Обсуждение

Значение cookie не доступно в массиве `$_COOKIE` в пределах того самого запроса, в котором cookie установлен. Другими словами, функция `setcookie()` не изменяет значения массива `$_COOKIE`. Однако при всех последующих запросах каждый установленный ранее cookie помещается

в массив $_COOKIE. Кроме того, если опция register_globals установлена в on, то значение cookie присваивается глобальной переменной.

Когда броузер посылает cookie обратно на сервер, то он посылает только значение. Невозможно получить доступ к домену, пути, времени истечения срока действия или статусу безопасности cookie через массив $_COOKIE, поскольку броузер не посылает его серверу.

Чтобы вывести на печать имена и значения всех cookies, посланных в текущем запросе, выполните цикл по массиву $_COOKIE:

```
foreach ($_COOKIE as $cookie_name => $cookie_value) {
    print "$cookie_name = $cookie_value<br>";
}
```

См. также

Рецепт 8.1, где показано, как устанавливать cookies; рецепт 8.3, демонстрирующий, как удалять cookies; рецепт 8.12, объясняющий буферизацию выходной информации; рецепт 8.18, показывающий, как избежать сообщения об ошибке «headers already sent», которое иногда появляется при вызове функции setcookie(); информацию об опции register_globals в рецепте 9.7.

8.3. Удаление cookies

Задача

Необходимо удалить cookie, так чтобы броузер не посылал его обратно серверу.

Решение

Вызовите функцию setcookie() с пустым значением для cookie и с временем истечения срока действия, соответствующим моменту времени в прошлом:

```
setcookie('flavor','',time()-86400);
```

Обсуждение

Неплохо установить время истечения срока действия на несколько часов или даже целый день назад, на тот случай, если часы сервера и часы пользовательского компьютера не синхронизированы. Например, если сервер «думает», что сейчас 3:02 P.M., то cookie с временем истечения срока действия, равным 3:05 P.M., не будет удален пользовательским компьютером, даже если время на сервере прошло.

Вызов функции setcookie(), удаляющий cookie, должен иметь те же самые аргументы (за исключением значения и времени), что и вызов функции setcookie(), который устанавливал cookie, поэтому включите путь, домен и флаг безопасности в случае необходимости.

См. также

Рецепт 8.1, где показано, как устанавливать cookies; рецепт 8.2, демонстрирующий, как читать значения cookie; рецепт 8.12, объясняющий буферизацию выходной информации; рецепт 8.18, показывающий, как избежать сообщения об ошибке «headers already sent», которое иногда появляется при вызове функции `setcookie()`; документацию по функции `setcookie()` на *http://www.php.net/setcookie*.

8.4. Перенаправление по другому адресу

Задача

Необходимо автоматически направить пользователя по новому URL. Например, требуется после успешной записи информации из формы перенаправить пользователя на страницу подтверждения данных.

Решение

Прежде чем сделать какой-нибудь вывод, вызовите функцию `header()`, чтобы послать заголовок `Location` с новым URL:

```
header('Location: http://www.example.com/');
```

Обсуждение

Для того чтобы передать переменные новой странице, можно включить их в строку запроса URL:

```
header('Location: http://www.example.com/?monkey=turtle');
```

URL, по которому перенаправляется пользователь, запрашивается броузером с помощью метода GET. Нельзя перенаправить кого-либо, чтобы он запросил страницу методом POST. Однако можно послать другие заголовки вместе с `Location`. Это особенно полезно в случае заголовка `Window-target`, который означает фрейм с определенным именем или окно, в которое надо загрузить новый URL:

```
header('Window-target: main');
header('Location: http://www.example.com/');
```

Перенаправляющий URL должен включать протокол и имя хоста; он не может представлять собой просто путь:

```
// Корректное перенаправление
header('Location: http://www.example.com/catalog/food/pemmican.php');

// Некорректное перенаправление
header('Location: /catalog/food/pemmican.php');
```

См. также

Документацию по функции `header()` на *http://www.php.net/header*.

8.5. Отслеживание сеанса работы с сайтом

Задача

Необходимо сохранять информацию о пользователе во время его путешествия по сайту.

Решение

Для этого предназначен модуль сеанса. Функция `session_start()` инициализирует сеанс, а заведение элемента в глобальном массиве `$_SESSION` указывает PHP, чтобы он отслеживал соответствующую переменную.

```
session_start();
$_SESSION['visits']++;
print 'You have visited here '.$_SESSION['visits'].' times.';
```

Обсуждение

Для того чтобы сеанс начинался автоматически при каждом обращении к сценариям сайта, надо установить опцию `session.auto_start` в 1 в файле *php.ini*. При этом отпадает необходимость в вызове функции `session_start()`.

Функции сеанса отслеживают пользователей, посылая им cookies со случайно сгенерированным идентификатором сеанса. Если PHP определяет, что пользователь не принимает cookie с идентификатором сеанса, то он автоматически добавляет идентификатор в URL и формы.[1,2] Например, рассмотрим следующий код, который печатает URL:

```
print '<a href="train.php">Take the A Train</a>';
```

Если сеансы разрешены, но пользователь не принимает cookies, то строка, посылаемая броузеру, может выглядеть примерно так:

```
<a href="train.php?PHPSESSID=2eb89f3344520d11969a79aea6bd2fdd">Take the A
Train</a>
```

В этом примере имя сеанса `PHPSESSID`, а идентификатор сеанса равен `2eb89f3344520d11969a79aea6bd2fdd`. PHP добавляет их к URL, и они вместе передаются следующей странице. Формы модифицируются так,

---

[1] До появления версии PHP 4.2.0 такое поведение разрешалось, только если PHP компилировался с опцией `--enable-trans-sid`.

[2] На самом деле все происходит наоборот: сначала PHP добавляет идентификатор сеанса и в cookie, и к URL/формам, а затем, если в URL идентификатор вернулся, а в cookie – нет, то таким образом PHP определяет, что cookies не поддерживаются. В любом случае такое поведение возможно, только если параметр `session.use_trans_sid` в *php.ini* установлен в `1`. В противном случае сеансы просто не будут работать у пользователей с отключенными cookies. – *Примеч. науч. ред.*

чтобы включить скрытые элементы, которые передают идентификатор сеанса. Перенаправления в заголовках Location автоматически не модифицируются, поэтому необходимо добавить к ним идентификатор сеанса с помощью константы SID:

```
$redirect_url = 'http://www.example.com/airplane.php';
if (defined('SID') && (! isset($_COOKIE[session_name()]))) {
    $redirect_url .= '?' . SID;
}

header("Location: $redirect_url");
```

Функция session_name() возвращает имя cookie, в котором хранится идентификатор сеанса, поэтому данный код добавляет константу SID к переменной $redirect_url, только если константа определена, а cookie сеанса не установлен.

По умолчанию PHP хранит данные сеанса в файлах в каталоге */tmp* на сервере. Каждый сеанс хранится в своем собственном файле. Для того чтобы изменить каталог, в который записываются файлы, установите значение конфигурационной опции session.save_path в файле *php.ini*, соответствующее новому каталогу. Для изменения каталогов можно также вызвать функцию session_save_path() с новым каталогом, но это надо делать перед обращением к любой переменной сеанса.

См. также

Документацию по функции session_start() на *http://www.php.net/session-start*, по функции session_save_path() на *http://www.php.net/session-save-path*; раздел «Sessions» в руководстве на *http://www.php.net/session*, описывающий модуль сеансов и его многочисленные параметры конфигурации, которые помогают управлять длительностью сеанса или способом их кэширования.

8.6. Хранение сеансов в базе данных

Задача

Необходимо хранить данные сеанса не в файле, а в базе данных. Если несколько веб-серверов используют одну и ту же базу данных, то данные сеанса доступны на всех этих веб-серверах.

Решение

Установите опцию session.save_handler в значение user в файле *php.ini* и используйте класс pc_DB_Session, показанный в примере 8.1. Например:

```
$s = new pc_DB_Session('mysql://user:password@localhost/db');
ini_get('session.auto_start') or session_start();
```

Обсуждение

Одна из наиболее сильных сторон модуля сеанса состоит в том, что он способен абстрагировать способ сохранения сеансов. Функция session_set_save_handler() указывает PHP, что различные операции с сеансами, такие как запись сеанса и чтение данных сеанса, следует организовывать посредством разных функций. Класс pc_DB_Session хранит информацию о сеансе в базе данных. Если с этой базой данных совместно работают несколько веб-серверов, то пользовательская информация о сеансе становится доступной на всех этих веб-серверах. Поэтому, если есть группа серверов, находящихся за узлом, регулирующим загрузку, то нет необходимости в каких-либо хитрых уловках для обеспечения корректности пользовательских данных о сеансе, независимо от того, какому серверу они были посланы.

Чтобы использовать pc_DB_Session, передайте имя источника данных (DSN, data source name) классу при его реализации. Данные сеанса сохраняются в таблице с именем php_session, которая имеет следующую структуру:

```
CREATE TABLE php_session (
  id CHAR(32) NOT NULL,
  data MEDIUMBLOB,
  last_access INT UNSIGNED NOT NULL,
  PRIMARY KEY(id)
)
```

Если требуется имя таблицы, отличное от php_session, установите значение опции session.save_path в файле *php.ini* в соответствии с новым именем таблицы. Пример 8.1 демонстрирует класс pc_DB_Session.

Пример 8.1. Класс pc_DB_Session

```
require 'PEAR.php';
require 'DB.php';

class pc_DB_Session extends PEAR {

    var $_dbh;
    var $_table;
    var $_connected = false;
    var $_gc_maxlifetime;
    var $_prh_read;
    var $error = null;

    /**
     * Конструктор
     */
    function pc_DB_Session($dsn = null) {
        if (is_null($dsn)) {
            $this->error = PEAR::raiseError('No DSN specified');
            return;
        }
```

```
        $this->_gc_maxlifetime = ini_get('session.gc_maxlifetime');
        // Сеанс продолжается один день, если не определено другое
        if (! $this->_gc_maxlifetime) {
            $this->_gc_maxlifetime = 86400;
        }

        $this->_table = ini_get('session.save_path');
        if ((! $this->_table) || ('/tmp' == $this->_table)) {
            $this->_table = 'php_session';
        }

        $this->_dbh = DB::connect($dsn);
        if (DB::isError($this->_dbh)) {
            $this->error = $this->_dbh;
            return;
        }

        $this->_prh_read = $this->_dbh->prepare(
            "SELECT data FROM $this->_table WHERE id LIKE ?
                                    AND last_access >= ?");
        if (DB::isError($this->_prh_read)) {
            $this->error = $this->_prh_read;
            return;
        }

        if (! session_set_save_handler(array(&$this,'_open'),
                                       array(&$this,'_close'),
                                       array(&$this,'_read'),
                                       array(&$this,'_write'),
                                       array(&$this,'_destroy'),
                                       array(&$this,'_gc'))) {
            $this->error = PEAR::raiseError('session_set_save_handler()
                                            failed');
            return;
        }

        return $this->_connected = true;
    }

    function _open() {
        return $this->_connected;
    }

    function _close() {
        return $this->_connected;
    }

    function _read($id) {
        if (! $this->_connected) { return false; }
        $sth =
            $this->_dbh->execute($this->_prh_read,
                                 array($id,time() - $this->_gc_maxlifetime));
        if (DB::isError($sth)) {
            $this->error = $sth;
```

```
            return '';
        } else {
            if (($sth->numRows() == 1) &&
                ($ar = $sth->fetchRow(DB_FETCHMODE_ORDERED))) {
                return $ar[0];
            } else {
                return '';
            }
        }
    }

    function _write($id,$data) {
        $sth = $this->_dbh->query(
          "REPLACE INTO $this->_table (id,data,last_access) VALUES (?,?,?)",
            array($id,$data,time()));
        if (DB::isError($sth)) {
            $this->error = $sth;
            return false;
        } else {
            return true;
        }
    }

    function _destroy($id) {
        $sth = $this->_dbh->query("DELETE FROM $this->_table WHERE id LIKE ?",
                                  array($id));
        if (DB::isError($sth)) {
            $this->error = $sth;
            return false;
        } else {
            return true;
        }
    }

    function _gc($maxlifetime) {
        $sth = $this->_dbh->query("DELETE FROM $this->_table
                                   WHERE last_access < ?",
                                  array(time() - $maxlifetime));
        if (DB::isError($sth)) {
            $this->error = $sth;
            return false;
        } else {
            return true;
        }
    }
}
```

Метод `pc_DB_Session::_write()` использует MySQL-специфическую команду SQL, `REPLACE INTO`, которая обновляет существующую запись или вставляет новую, в зависимости от того, есть ли уже в базе данных запись с этим идентификатором поля. Если вы используете другую базу данных, модифицируйте функцию `write()` для выполнения той же са-

мой задачи. Например, удаляем существующую строку (если она есть) и вставляем новую, выполняя все это в одной транзакции:

```
function _write($id,$data) {
    $sth = $this->_dbh->query('BEGIN WORK');
    if (DB::isError($sth)) {
        $this->error = $sth;
        return false;
    }
    $sth = $this->_dbh->query("DELETE FROM $this->_table WHERE id LIKE ?",
                              array($id));
    if (DB::isError($sth)) {
        $this->error = $sth;
        $this->_dbh->query('ROLLBACK');
        return false;
    }
    $sth = $this->_dbh->query(
       "INSERT INTO $this->_table (id,data,last_access) VALUES (?,?,?)",
        array($id,$data,time()));
    if (DB::isError($sth)) {
        $this->error = $sth;
        $this->_dbh->query('ROLLBACK');
        return false;
    }
    $sth = $this->_dbh->query('COMMIT');
    if (DB::isError($sth)) {
        $this->error = $sth;
        $this->_dbh->query('ROLLBACK');
        return false;
    }
         return true;
}
```

См. также

Документацию по функции `session_set_save_handler()` на *http://www.php.net/session-set-save-handler*; информацию об обработчике, использующем PostgreSQL, на *http://www.zend.com/codex.php?id=456&single=1*; рецепт 10.3, в котором рассматривается формат имен источника данных.

8.7. Идентификация различных броузеров

Задача

Необходимо сгенерировать содержимое, основанное на возможностях броузера пользователя.

Решение

Характеристики броузера можно определить с помощью объекта, возвращенного функцией `get_browser()`:

```
$browser = get_browser();

if ($browser->frames) {
    // вывод формата на основе фрейма
} elseif ($browser->tables) {
    // вывод формата на основе таблицы
} else {
    // вывод монотонного формата
}
```

Обсуждение

Функция `get_browser()` проверяет переменную окружения `$_ENV['HTTP_USER_AGENT']` (установленную веб-сервером) и сравнивает ее с броузерами, перечисленными в файле характеристик броузеров. Из-за проблем с лицензированием PHP поставляется без файла характеристик броузеров. В разделе «Obtaining PHP» секции FAQ (на *http://www.php.net/faq.obtaining*) как источники файла характеристик броузера указаны адреса *http://www.cyscape.com/asp/browscap/* и *http://www.amrein.com/apps/page.asp?Q=InowDownload*, но есть еще один на *http://asp.net.do/browscap.zip*.[1]

Загрузив файл характеристик броузеров, необходимо указать PHP, где его найти, прописав в параметре конфигурациии `browscap` соответствующий путь к файлу. Если PHP используется в качестве CGI, установите следующий параметр в файле *php.ini*:

```
browscap=/usr/local/lib/browscap.txt
```

Если используется Apache, то необходимо установить параметр в конфигурационном файле Apache:

```
php_value browscap "/usr/local/lib/browscap.txt"
```

Многие характеристики, которые может определить функция `get_browser()`, показаны в табл. 8.1. Хотя для характеристик, определяемых пользователем, таких как `javascript` или `cookies`, функция `get_browser()` лишь сообщает, способен ли броузер поддерживать эти возможности. Она не ничего не сообщает, если пользователь запретил эти функции броузера. Если JavaScript выключена в броузере, который поддерживает JavaScript, или пользователь отказывается принимать cookies, когда броузер запрашивает его, то функция `get_browser()` все равно сообщает, что броузер поддерживает эти функции.

---

[1] И еще: *http://www.garykeith.com/browsers/downloads.asp*. – *Примеч. ред.*

Таблица 8.1. Свойства объекта, показывающие характеристики броузера

Свойство	Описание
platform	Операционная система, в которой запущен броузер (т. е. Windows, Macintosh, UNIX, Win32, Linux, MacPPC)
version	Полная версия броузера (например, 5.0, 3.5, 6.0b2)
majorver	Старшая часть версии броузера (например, 5, 3, 6)
minorver	Младшая часть версии броузера (например, 0, 5, 02)
frames	1, если броузер поддерживает фреймы
tables	1, если броузер поддерживает таблицы
cookies	1, если броузер поддерживает cookies
backgroundsounds	1, если броузер поддерживает фоновые звуки с помощью тегов <embed> или <bgsound>
vbscript	1, если броузер поддерживает VBScript
javascript	1, если броузер поддерживает JavaScript
javaapplets	1, если броузер может запускать Java-апплеты
activexcontrols	1, если броузер может запускать элементы управления ActiveX

См. также

Документация по функции get_browser() на *http://www.php.net/get-browser*.

8.8. Формирование строки запроса GET

Задача

Необходимо сформировать ссылку, которая содержит пары имя/значение в строке запроса.

Решение

Закодируйте имена и значения с помощью функции urlencode(), а строку запроса создайте посредством функции join():

```
$vars = array('name' => 'Oscar the Grouch',
              'color' => 'green',
              'favorite_punctuation' => '#');
$safe_vars = array();
foreach ($vars as $name => $value) {
    $safe_vars[] = urlencode($name).'='.urlencode($value);
}

$url = '/muppet/select.php?' . join('&',$safe_vars);
```

Обсуждение

URL, сформированный в данном решении, выглядит следующим образом:

```
/muppet/select.php?name=Oscar+the+Grouch&color=green&favorite_punctuation=%23
```

Строка запроса содержит пробелы, закодированные символом +. Специальные символы, такие как #, записаны в шестнадцатеричной кодировке, например %23, поскольку ASCII-значение символа # равно 35, что эквивалентно 23 в шестнадцатеричном коде.

Несмотря на то что функция urlencode() предотвращает появление любых специальных символов в именах или значениях переменных, получаемых после разбора сконструированного URL, можно столкнуться с проблемами, если имена переменных начинаются с названий примитивов HTML. Рассмотрим следующий неполный URL, предназначенный для получения информации о стереосистеме:

```
/stereo.php?speakers=12&cdplayer=52&amp=10
```

Элементом HTML, отображающим амперсанд (&), является & поэтому броузер может интерпретировать этот URL как:

```
/stereo.php?speakers=12&cdplayer=52&=10
```

Есть три способа избежать искажения URL, которое могут вызвать внедренные элементы. Первый заключается в том, чтобы выбирать имена переменных, которые нельзя спутать с примитивами, например, _amp вместо amp. А второй – в преобразовании символов с эквивалентами примитивов HTML в эти элементы перед выводом URL. Последнее делается с помощью функции htmlentities():

```
$url = '/muppet/select.php?' . htmlentities(join('&',$safe_vars));
```

В результате URL будет выглядеть так:

```
/muppet/select.php?name=Oscar+the+Grouch&color=green&favorite_punctuation=%23
```

Третий способ состоит в том, чтобы заменить значение разделителя аргументов & на ; путем установки параметра конфигурации arg_separator.input в ;. Затем надо объединить пары имя-значение с символом ; для получения строки запроса:

```
/muppet/select.php?name=Oscar+the+Grouch;color=green;favorite_punctuation=%23
```

Трудности могут возникнуть с любым методом GET в URL, в котором нельзя явным образом указывать точку с запятой, например в форме с методом GET, поскольку пользовательский броузер воспринимает символ & в качестве аргумента-разделителя.

Многие броузеры не поддерживают использование символа ; в качестве аргумента-разделителя, поэтому проще всего избежать проблем с элементами в URL, выбирая имена переменных, которые не совпада-

ют с именами элементов. Если вы не можете полностью управлять именами переменных, то защитите URL от декодирования элементов с помощью функции `htmlentities()`.[1]

См. также

Документацию по функции `urlencode()` на *http://www.php.net/urlencode* и по функции `htmlentities()` на *http:/ /www.php.net/htmlentities*.

8.9. Применение базовой аутентификации HTTP

Задача

Необходимо использовать PHP для защиты разделов веб-сайта с помощью паролей. Вместо того чтобы хранить пароли во внешнем файле и возлагать на сервер функции проверки регистрационной информации пользователей, надо реализовать логику проверки паролей в PHP-программе.

Решение

Глобальные переменные `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW']` хранят предоставленные пользователем имя и пароль. Для того чтобы закрыть доступ к странице, пошлите заголовок `WWW-Authenticate`, обозначающий область (realm) аутентификации, вместе с кодом состояния 401:

```
header('WWW-Authenticate: Basic realm="My Website"');
header('HTTP/1.0 401 Unauthorized');
echo "You need to enter a valid username and password.";
exit;
```

Обсуждение

Увидев заголовок 401, броузер показывает диалоговое окно для ввода имени пользователя и пароля. Если это удостоверение личности (имя пользователя и пароль) принимается сервером, то оно ассоциируется с областью, указанной в заголовке `WWW-Authenticate`. Код, который проверяет удостоверение личности (credentials), должен быть выполнен раньше, чем будет послан какой-либо вывод в броузер, т. к. может быть послан заголовок. Например, это можно сделать с помощью функции, подобной `pc_validate()`, которая показана в примере 8.2.

---

[1] Надо сказать, что пример несколько надуманный, т. к. в нем рассматривается строка `&amp` без завершающей точки с запятой, являющейся обязательным элементом сущности HTML. – *Примеч. науч. ред.*

Пример 8.2. pc_validate( )

```
function pc_validate($user,$pass) {
    /* замените на соответствующую проверку имени пользователя и пароля,
       например на проверку в базе данных */
    $users = array('david' => 'fadj&32',
                   'adam'  => '8HEj838');

    if (isset($users[$user]) && ($users[$user] == $pass)) {
        return true;
    } else {
        return false;
    }
}
```

Ниже приведен пример применения функции `pc_validate()`:

```
if (! pc_validate($_SERVER['PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW'])) {
    header('WWW-Authenticate: Basic realm="My Website"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You need to enter a valid username and password.";
    exit;
}
```

Замените содержание функции `pc_validate()` на вашу собственную логику проверки правильности ввода пароля пользователем. Можно также заменить строку области «My Website», а также сообщение «You need to enter a valid username and password», появляющееся при нажатии пользователем кнопки «cancel» в диалоговом окне аутентификации его броузера.

Нельзя применять базовую аутентификацию HTTP, если PHP используется в качестве CGI. Если нельзя исполнять PHP как серверный модуль, то можно обратиться к аутентификации на основе cookies, рассмотренной в рецепте 8.10.

Другая особенность базовой аутентификации HTTP заключается в том, что она не предоставляет другого способа выхода пользователя из сеанса, кроме закрытия броузера. Руководство по PHP на *http://www.php.net/features.http-auth* предлагает некоторые способы выхода пользователя, работающие с различной степенью успеха на разных комбинациях серверов и броузеров.

Однако есть прямой путь заставить пользователя завершить работу по истечении фиксированного интервала времени: включить вычисление времени в строку области. Броузеры проверяют одну и ту же комбинацию пользовательского имени и пароля каждый раз, когда они запрашивают удостоверение личности в той же самой области. Изменение имени области принуждает броузер запрашивать у пользователя новые идентификационные данные. Например, следующий код вызывает принудительный выход каждую полночь:

```
if (! pc_validate($_SERVER['PHP_AUTH_USER'],$_SERVER['PHP_AUTH_PW'])) {
    $realm = 'My Website for '.date('Y-m-d');
```

```
    header('WWW-Authenticate: Basic realm="'.$realm.'"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You need to enter a valid username and password.";
    exit;
}
```

Можно также определить время выхода для каждого пользователя, не изменяя при этом имя области, путем сохранения времени регистрации пользователя или времени его доступа к защищенной странице. Функция `pc_validate2()` записывает время регистрации в базу данных и вызывает принудительный выход, если прошло более 15 минут с момента последней регистрации пользователя на защищенной странице.

Пример 8.3. pc_validate2( )

```
Function pc_validate2($user,$pass) {
    $safe_user = strtr(addslashes($user),array('_' => '\_', '%' => '\%'));
    $r = mysql_query("SELECT password,last_access
                      FROM users WHERE user LIKE '$safe_user'");

    if (mysql_numrows($r) == 1) {
        $ob = mysql_fetch_object($r);
        if ($ob->password == $pass) {
            $now = time();
            if (($now - $ob->last_access) > (15 * 60)) {
                return false;
            } else {
                // обновление времени последнего доступа
                mysql_query("UPDATE users SET last_access = NOW()
                             WHERE user LIKE '$safe_user'");
               return true;
            }
        }
    } else {
        return false;
    }
}
```

Например:

```
if (! pc_validate($_SERVER['PHP_AUTH_USER'],$_SERVER['PHP_AUTH_PW'])) {
    header('WWW-Authenticate: Basic realm="My Website"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You need to enter a valid username and password.";
    exit;
}
```

См. также

Рецепт 8.10; раздел «HTTP authentication with PHP» (Аутентификация HTTP средствами PHP) в руководстве по PHP на *http://www.php.net/features.http-auth*.

8.10. Аутентификация, основанная на cookies

Задача

Необходим больший контроль над процедурой авторизации пользователя, например разработка вашей собственной формы авторизации.

Решение

Сохраните статус аутентификации в cookie или сеансе. В случае успешной регистрации пользователя поместите его имя в cookie. А также поместите хеш из имени пользователя и секретного слова, так чтобы пользователь не смог сконструировать cookie с именем пользователя в нем:

```
$secret_word = 'if i ate spinach';
if (pc_validate($_REQUEST['username'],$_REQUEST['password'])) {
    setcookie('login',

$_REQUEST['username'].',.md5($_REQUEST['username'].$secret_word));
}
```

Обсуждение

При использовании этого способа аутентификации необходимо вызывать собственную форму авторизации:

```
<form method="post" action="login.php">
Username: <input type="text" name="username"> <br>
Password: <input type="password" name="password"> <br>
<input type="submit" value="Log In">
</form>
```

Для проверки имени пользователя и пароля можно применять ту же самую функцию `pc_validate()` из рецепта 8.9. Единственное различие состоит в том, что в качестве удостоверения личности ей передаются `$_REQUEST['username']` и `$_REQUEST['password']` вместо `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW']`. После проверки пароля установите cookie, который содержит имя пользователя и хеш имени пользователя и секретного слова. Этот хеш помешает пользователю подделать авторизацию простой посылкой cookie, содержащего имя пользователя.

После того как пользователь прошел авторизацию, сценарий должен лишь удостовериться, что был послан корректный авторизационный cookie, и выполнить некоторые специальные действия для данного зарегистрированного пользователя:

```
unset($username);
if ($_COOKIE['login']) {
    list($c_username,$cookie_hash) = split(',',$_COOKIE['login']);
    if (md5($c_username.$secret_word) == $cookie_hash) {
```

```
        $username = $c_username;
    } else {
        print "You have sent a bad cookie.";
    }
}

if ($username) {
    print "Welcome, $username.";
} else {
    print "Welcome, anonymous user.";
}
```

Если применяются встроенные средства поддержки сеанса, то можно добавить имя пользователя и хеш к сеансу, чтобы не посылать отдельный cookie. Если пользователь авторизуется, установите дополнительную переменную сеанса, вместо того чтобы посылать cookie:

```
if (pc_validate($_REQUEST['username'],$_REQUEST['password'])) {
    $_SESSION['login'] =
        $_REQUEST['username'].','.md5($_REQUEST['username'].$secret_word));
}
```

Код проверки практически тот же самый, только в нем вместо массива `$_COOKIE` используется массив `$_SESSION`:

```
unset($username);
if ($_SESSION['login']) {
    list($c_username,$cookie_hash) = explode(',',$_SESSION['login']);
    if (md5($c_username.$secret_word) == $cookie_hash) {
        $username = $c_username;
    } else {
        print "You have tampered with your session.";
    }
}
```

Применение cookie или сеансовой аутентификации вместо базовой аутентификации HTTP позволяет упростить выход пользователя – достаточно удалить его регистрационный cookie или переменную регистрации из его сеанса. Другое преимущество хранения информации об авторизации в сеансе заключается в том, что можно сравнить перемещения пользователей по сайту в то время, когда они авторизованы, с их перемещениями до и после авторизации. В случае базовой аутентификации HTTP не существует способа ассоциировать запросы страниц сайта, сделанные некоторым пользователем, с запросами того же пользователя, сделанными до того, как он авторизовался. Попытка привязать имя пользователя к IP-адресу будет ошибочной, особенно если пользовательский компьютер находится за брандмауэром или прокси-сервером. При использовании сеанса можно изменить процедуру регистрации с целью фиксации связи между идентификатором сеанса и именем пользователя:

```
if (pc_validate($_REQUEST['username'],$_REQUEST['password'])) {
    $_SESSION['login'] =
```

```
        $_REQUEST['username'].',.md5($_REQUEST['username'].$secret_word));
    error_log('Session id '.session_id().' log in as
'.$_REQUEST['username']);
}
```

В приведенном выше примере сообщение записывается в журнал ошибок, но с таким же успехом эту информацию можно записать в базу данных, если она используется для анализа работы сайта или трафика.

Одной из опасностей, связанных с применением идентификатора сеанса, является возможность его похищения. Если Алиса угадает идентификатор сеанса Боба, то она сможет замаскироваться под него на веб-сервере. Модуль сеанса имеет два необязательных параметра конфигурации, затрудняющих угадывание идентификатора сеанса. Параметр `session.entropy_file` содержит путь к устройству или файлу, которые обеспечивают элемент случайности, например */dev/random* или */dev/urandom*. Параметр `session.entropy_length` содержит количество бит, которые надо прочитать из статистического файла при создании идентификатора сеанса.

Не имеет значения, насколько тяжело угадать идентификаторы сеанса, т. к. они могут быть украдены, если пересылаются между сервером и броузером пользователя открытым текстом. Базовая аутентификация HTTP также имеет этот недостаток. Для защиты от просмотра сетевого трафика следует применять SSL, как описано в рецепте 14.10.

См. также

Рецепт 8.9; рецепт 8.17, в котором обсуждаются ошибки авторизации; рецепт 14.3, в котором рассматривается проверка данных с использованием хешей; документацию по функции `setcookie()` на *http://www.php.net/setcookie* и по функции `md5()` на *http://www.php.net/md5*.

8.11. Передача выходной информации в броузер

Задача

Необходимо немедленно послать выходную информацию в броузер. Например, если вы хотите сообщить пользователю о выполнении медленного запроса к базе данных.

Решение

Это делается при помощи функции `flush()`:

```
print 'Finding identical snowflakes...';
flush();
$sth = $dbh->query(
    'SELECT shape,COUNT(*) AS c FROM snowflakes GROUP BY shape HAVING c > 1');
```

Обсуждение

Функция `flush()` посылает веб-серверу весь вывод, который PHP буферизировал, но веб-сервер может обладать своим собственным буфером, что вызывает задержку, когда информация передается броузеру. Кроме того, некоторые броузеры не показывают информацию сразу по принятии, а некоторые версии броузера Internet Explorer (IE) не показывают страницу, пока не будут приняты хотя бы 256 байт. Чтобы заставить IE показывать содержание, напечатайте пробелы в начале страницы:

```
print str_repeat(' ',300);
print 'Finding identical snowflakes...';
flush();
$sth = $dbh->query(
    'SELECT shape,COUNT(*) AS c FROM snowflakes GROUP BY shape HAVING c > 1');
```

См. также

Рецепт 18.17; документацию по функции `flush()` на *http://www.php.net/flush*.

8.12. Буферизация вывода в броузер

Задача

Необходимо начать генерацию вывода раньше, чем закончится пересылка заголовков или cookies.

Решение

Вызовите функцию `ob_start()` в начале вашей страницы и функцию `ob_end_flush()` в конце страницы. Кроме того, можно поменять местами команды, генерирующие код, и команды, посылающие заголовки. Выходная информация не будет послана до тех пор, пока не будет вызвана функция `ob_end_flush()`:

```
<?php ob_start(); ?>

I haven't decided if I want to send a cookie yet.

<?php setcookie('heron','great blue'); ?>

Yes, sending that cookie was the right decision.

<?php ob_end_flush(); ?>
```

Обсуждение

Можно передать функции `ob_start()` имя функции обратного вызова (callback), чтобы она обработала буфер вывода. Это полезно для повторной обработки всего содержимого страницы, например чтобы скрыть адреса электронной почты от автоматов, отыскивающих адреса:

```
<?php
function mangle_email($s) {
    return preg_replace('/([^@\s]+)@([-a-z0-9]+\.)+[a-z]{2,}/is',
                        '<$1@...>',
                        $s);
}

ob_start('mangle_email');
?>

I would not like spam sent to ronald@example.com!

<?php ob_end_flush(); ?>
```

Функция `mangle_email()` преобразует вывод в:

```
I would not like spam sent to <ronald@...>!
```

Параметр конфигурации `output_buffering` включает буферизацию вывода для всех страниц:

```
output_buffering = On
```

Точно так же параметр `output_handler` устанавливает функцию обратного вызова для обработки буфера вывода, которая будет использована на всех страницах:

```
output_handler=mangle_email
```

Установка параметра `output_handler` автоматически устанавливает параметр `output_buffering` в `on`.

См. также

Рецепт 10.10, в котором рассмотрена буферизация вывода в функции регистрации ошибок базы данных; документацию по функции `ob_start()` на *http://www.php.net/ob-start*, по функции `ob_end_flush()` на *http://www.php.net/ob-end-flush* и по буферизации вывода на *http://www.php.net/outcontrol*.

8.13. Сжатие веб-вывода с помощью gzip

Задача

Необходимо посылать сжатую информацию в броузер, поддерживающий автоматическую декомпрессию.

Решение

Добавьте следующую настройку в ваш файл *php.ini*:

```
zlib.output_compression=1
```

Обсуждение

Броузеры сообщают серверу о том, что они могут принимать сжатые ответы с помощью заголовка `Accept-Encoding`. Если броузер посылает `Accept-Encoding: gzip` или `Accept-Encoding: deflate`, а PHP скомпилирован с расширением *zlib*, то параметр конфигурации `zlib.output_compression` приказывает PHP сжать вывод с помощью соответствующего алгоритма перед возвращением его броузеру. Броузер распаковывает данные перед их показом.

Можно установить уровень сжатия с помощью параметра конфигурации `zlib.output_compression_level`:

```
; minimal compression
(минимальное сжатие)
zlib.output_compression_level=1

; maximal compression
(максимальное сжатие)
zlib.output_compression_level=9
```

Если задан максимальный уровень сжатия, серверу приходится посылать меньше данных броузеру, но при этом сервер тратит больше времени на сжатие данных.

См. также

Документацию по расширению *zlib* на *http://www.php.net/zlib*.

8.14. Сокрытие от пользователей сообщений об ошибках

Задача

Необходимо скрыть от пользователей сообщения PHP об ошибках.

Решение

Установите следующие значения в вашем файле *php.ini* или в конфигурационном файле веб-сервера:

```
display_errors =off
log_errors     =on
```

Руководствуясь этими настройками, PHP не отображает ошибки в броузере в виде HTML, а сохраняет их в серверном журнале ошибок.

Обсуждение

Если параметр конфигурации `log_errors` установлен в `on`, то сообщения об ошибках записываются в серверный журнал ошибок. Для того что-

бы PHP записывал ошибки в отдельный файл, присвойте параметру `error_log` имя этого файла:

```
error_log   = /var/log/php.error.log
```

Если параметр `error_log` установлен в значение `syslog`, то сообщения PHP об ошибках посылаются системному регистратору ошибок с помощью *syslog(3)* в UNIX и Event Log в Windows NT.

Существует множество сообщений об ошибках, которые следует показывать пользователю, например о том, что он неправильно заполнил форму, но необходимо оградить пользователя от внутренних ошибок, которые могут быть результатом недостатков программы. Для этого есть две причины. Во-первых, эти ошибки показывают ваш непрофессионализм (опытным пользователям) и, во-вторых, вносят путаницу (для новичков). Если что-то идет не так во время записи входной информации формы в базу данных, то надо проверить код возврата в ответ на запрос в базу данных и послать сообщение пользователю с извинениями и просьбой зайти попозже. Показ пользователям малопонятных сообщений непосредственно от PHP не способствует повышению доверия к вашему сайту.

Во-вторых, показ пользователю сообщений об ошибках представляет риск для безопасности. В зависимости от базы данных и типа ошибок, сообщения об ошибках могут содержать информацию о способе регистрации в базе данных и о ее структуре. Злоумышленник, воспользовавшись этой информацией, может предпринять атаку на ваш сайт.

Например, если сервер базы данных «упал», а вы пытаетесь соединиться с ним с помощью функции `mysql_connect()`, то PHP выдаст следующее предупреждение:

```
<br>
<b>Warning</b>:  Can't connect to MySQL server on 'db.example.com' (111) in
<b>/www/docroot/example.php</b> on line <b>3</b><br>
```

Пользователь, броузеру которого будет послано это предупреждающее сообщение, узнает, что ваш сервер базы данных называется *db.example.com*, и сможет организовать на него атаку.

См. также

Рецепт 8.17 о том, как регистрировать ошибки; документацию по параметрам конфигурации PHP на *http://www.php.net/configuration*.

8.15. Настройка обработки ошибок

Задача

Необходимо изменить уровень регистрации ошибок на определенной странице. Это позволяет управлять типом отображаемых ошибок.

Решение

Типы ошибок, на которые будет реагировать PHP, настраиваются с помощью функции `error_reporting()`:

```
error_reporting(E_ALL);                 // все
error_reporting(E_ERROR | E_PARSE);     // только основные проблемы
error_reporting(E_ALL & ~E_NOTICE);     // все, за исключением уведомлений
```

Обсуждение

Каждая сгенерированная ошибка имеет тип, к которому она относится. Например, если попытаться применить функцию `array_pop()` к строке, то PHP пожалуется, что «This argument needs to be an array» (Этот аргумент должен быть массивом), поскольку извлекать последний элемент можно только из массивов. С этим сообщением ассоциируется тип ошибки `E_NOTICE`, а не фатальная ошибка времени выполнения (runtime error).

По умолчанию уровень сообщений об ошибках равен `E_ALL & ~E_NOTICE`, что означает все типы ошибок, за исключением уведомлений. Символ `&` – это логическое `И`, а символ `~` – логическое `НЕТ`. Однако рекомендованный файл конфигурации *php.ini* устанавливает уровень сообщения об ошибках, равный `E_ALL`, что означает все типы ошибок.

Сообщения об ошибках, отмеченные как notice, это ошибки времени выполнения, но не такие серьезные, как предупреждения. Это не обязательно что-то неверное, но они говорят о потенциальной проблеме. Один из примеров ошибки типа `E_NOTICE` – «Undefined variable»; она случается при попытке использовать переменную без предварительно присвоенного значения:

```
// Генерирует E_NOTICE
foreach ($array as $value) {
    $html .= $value;
}

// Не генерирует никакого сообщения об ошибках
$html = '';
foreach ($array as $value) {
    $html .= $value;
}
```

В первом случае при начальном проходе цикла `foreach` переменная `$html` не определена. Поэтому когда ее содержимое складывается со значением, PHP сообщает о попытке сложения с неопределенной переменной. Во втором случае пустая строка присваивается переменной `$html` вне цикла, для того чтобы избежать сообщения `E_NOTICE`. Предыдущие два программные фрагмента порождают один и тот же код, поскольку по умолчанию значение переменной представляет собой пустую строку. Сообщение `E_NOTICE` может быть полезным, поскольку можно сделать, например, ошибку в имени переменной:

```
foreach ($array as $value) {
    $hmtl .= $value; // Ой! Это должна быть $html
}

$html = ''
foreach ($array as $value) {
    $hmtl .= $value; // Ой! Это должна быть $html
}
```

Пользовательская функция обработки ошибок может анализировать ошибки на основании их типа и предпринимать соответствующие действия. Полный список типов ошибок показан в табл. 8.2.

Таблица 8.2. Типы ошибок

Значение	Константа	Описание	Уловимая
1	E_ERROR	Неисправимая ошибка	Нет
2	E_WARNING	Исправимая ошибка	Да
4	E_PARSE	Синтаксическая ошибка	Нет
8	E_NOTICE	Вероятная ошибка	Да
16	E_CORE_ERROR	Подобная E_ERROR, но сгенерированная ядром PHP	Нет
32	E_CORE_ WARNING	Подобная E_WARNING, но сгенерированная ядром PHP	Нет
64	E_COMPILE_ ERROR	Подобная E_ERROR, но сгенерированная Zend Engine	Нет
128	E_COMPILE_ WARNING	Подобная E_WARNING, но сгенерированная Zend Engine	Нет
256	E_USER_ERROR	Подобная E_ERROR, но инициированная вызовом функции trigger_error()	Да
512	E_USER_ WARNING	Подобная E_WARNING, но инициированная вызовом функции trigger_error()	Да
1024	E_USER_NOTICE	Подобная E_NOTICE, но инициированная вызовом функции trigger_error()	Да
2047	E_ALL	Все	n/a

Ошибки, отмеченные как уловимые, могут быть обработаны функцией, зарегистрированной с помощью функции set_error_handler(). Остальные соответствуют настолько серьезным проблемам, что пользователю их обрабатывать небезопасно, поэтому о них должен позаботиться PHP.

См. также

Рецепт 8.16, объясняющий, как устанавливать пользовательский обработчик ошибок; документацию по функции `error_reporting()` на *http://www.php.net/error-reporting* и по функции `set_error_handler()` на *http://www.php.net/set-error-handler*; более подробную информацию об ошибках на *http://www.php.net/ref.errorfunc.php*.

8.16. Применение пользовательского обработчика ошибок

Задача

Необходимо создать пользовательский обработчик ошибок, позволяющий управлять уровнем сообщений PHP об ошибках.

Решение

Для установки собственной функции обработки ошибок применяется функция `set_error_handler()`:

```
set_error_handler('pc_error_handler');

function pc_error_handler($errno, $error, $file, $line) {
    $message = "[ERROR][$errno][$error][$file:$line]";
    error_log($message);
}
```

Обсуждение

Пользовательская функция обработки ошибок может анализировать ошибки на основании их типа и предпринимать соответствующие действия. Список типов ошибок приведен в табл. 8.2 рецепта 8.15.

Передайте функции `set_error_handler()` имя функции, и PHP будет направлять все ошибки этой функции. Функция обработки ошибок может принимать до пяти параметров. Первый параметр – это тип ошибки, например `8` для `E_NOTICE`. Второй параметр – это текст сообщения об ошибке, такой как «Undefined variable: html». Третий и четвертый аргументы содержат имя файла и номер строки, в которой PHP обнаружил ошибку. Последний параметр представляет массив, содержащий все переменные, определенные в текущей области видимости, и их значения.

Например, в следующем фрагменте кода к переменной `$html` прибавляется значение без предварительного присваивания начального значения:

```
error_reporting(E_ALL);
set_error_handler('pc_error_handler');

function pc_error_handler($errno, $error, $file, $line, $context) {
    $message = "[ERROR][$errno][$error][$file:$line]";
```

```
    print "$message";
    print_r($context);
}

$form = array('one','two');

foreach ($form as $line) {
    $html .= "<b>$line</b>";
}
```

Когда генерируется ошибка «Undefined variable», то функция `pc_error_handler()` печатает:

```
[ERROR][8][Undefined variable:  html][err-all.php:16]
```

Вслед за начальным сообщением об ошибке функция `pc_error_handler()` печатает большой массив, содержащий все глобальные переменные, переменные окружения, переменные запросов и переменные сеанса.

Еще раз подчеркнем, что ошибки, отмеченные в табл. 8.2 как уловимые, можно обработать функцией, зарегистрированной с помощью функции `set_error_handler()`. С остальными связаны настолько серьезным проблемы, что пользователю лучше передоверить их обработку PHP.

См. также

Рецепт 8.15, в котором перечислены различные типы ошибок; документацию по функции `set_error_handler()` на *http://www.php.net/set-error-handler*.

8.17. Регистрация ошибок

Задача

Необходимо записывать ошибки программы в журнал. Эти ошибки могут включать все – от синтаксических ошибок и ненайденных файлов до некорректных запросов в базу данных и потерянных соединений.

Решение

Для записи в журнал ошибок предназначена функция `error_log()`:

```
// Ошибка LDAP
if (ldap_errno($ldap)) {
    error_log("LDAP Error #" . ldap_errno($ldap) . ": " . ldap_error($ldap));
}
```

Обсуждение

Ведение журнала ошибок упрощает процесс отладки. Разумная регистрация ошибок облегчает их исправление. Всегда записывайте информацию о причине ошибки:

```
$r = mysql_query($sql);
if (! $r) {
    $error = mysql_error();
    error_log('[DB: query @'.$_SERVER['REQUEST_URI']."][$sql]: $error");
} else {
    // результаты обработки
}
```

Вы не получите необходимую помощь в процессе отладки, какую могли бы получить, если фиксируете лишь сам факт ошибки без какой-либо вспомогательной информации:

```
$r = mysql_query($sql);
if (! $r) {
    error_log("bad query");
} else {
    // результат обработки
}
```

Другим полезным приемом является включение констант `__FILE__` и `__LINE__` в сообщения об ошибках:

```
error_log('['.__FILE__.']['.__LINE__."]: $error");
```

Константа `__FILE__` – это текущее имя файла, а `__LINE__` – номер текущей строки.

См. также

Рецепт 8.14 о сокрытии сообщений об ошибках от пользователя; документацию по функции `error_log()` на *http://www.php.net/error-log*.

8.18. Устранение ошибок «headers already sent» (заголовки уже посланы)

Задача

При попытке послать заголовок HTTP или cookie с помощью функций `header()` или `setcookie()` PHP выдает сообщение об ошибке «headers already sent» (заголовки уже посланы).

Решение

Эта ошибка возникает при передаче выходной информации, отличной от заголовка, до вызова функций `header()` или `setcookie( )`.

Перепишите программу так, чтобы любой вывод осуществлялся после передачи заголовков:

```
// правильно
setcookie("name", $name);
print "Hello $name!";
```

```
// неправильно
print "Hello $name!";
setcookie("name", $name);

// правильно
<?php setcookie("name",$name); ?>
<html><title>Hello</title>
```

Обсуждение

Сообщения HTTP имеют заголовок и тело, пересылаемые пользователю именно в таком порядке. Если посылка тела начата, то уже нельзя послать какой-либо заголовок. Поэтому если функция `setcookie()` вызывается после печати некоторого содержания HTML, то PHP не может послать соответствующий заголовок `Cookie`.

Кроме того, необходимо удалять замыкающий пробельный символ в любом включаемом файле. Если включить файл с строками, содержащими пробельные смволы, вне тегов `<?php ?>`, то эти строки будут переданы броузеру. Для удаления ведущих и замыкающих пробельных строк из файла предназначена функция `trim()`:

```
$file = '/path/to/file.php';

// делаем резервную копию
   copy($file, "$file.bak") or die("Can't copy $file: $php_errormsg);

// читаем и удаляем концевые пробелы
$contents = trim(join('',file($file)));

// записываем
$fh = fopen($file, 'w')  or die("Can't open $file for writing: $php_errormsg);
if (-1 == fwrite($fh, $contents)) { die("Can't write to $file: $php_errormsg); }
fclose($fh)              or die("Can't close $file: $php_errormsg);
```

Возможно, вместо обработки файлов по принципу один за другим было бы удобнее обрабатывать их по принципу каталог за каталогом. О том, как обработать все файлы в каталоге, рассказано в рецепте 19.7.

Если вы не хотите беспокоиться о пробельных строках, нарушающих посылку заголовков, включите буферизацию вывода. Буферизация выходной информации не дает PHP немедленно посылать весь вывод клиенту, и в случае применения буфера вывода можно смело смешивать заголовки и тело. Однако пользователям может показаться, что ваш сервер стал тратить больше времени на выполнение их запросов, поскольку им приходится дольше ждать, когда броузер выведет на экран какую-нибудь информацию.

См. также

О буферизации вывода в рецепте 8.12; рецепт 19.7, в котором рассматривается обработка всех файлов в каталоге; документацию по функции `header()` на *http://www.php.net/header*.

8.19. Регистрация отладочной информации

Задача

Необходимо упростить отладку путем добавления в сценарий операторов вывода значений переменных. Но при этом требуется, чтобы была возможность легко переключаться между режимами выполнения и отладки.

Решение

Параметр конфигурации `auto_prepend_file` позволяет поместить на страницу функцию, которая в зависимости от установленных констант будет печатать сообщения. Сохраните следующий код в файле *debug.php*:

```
// включаем отладку
define('DEBUG',true);

// конструируем отладочную функцию
function pc_debug($message) {
    if (defined(DEBUG) && DEBUG) {
        error_log($message);
    }
}
```

Установите параметр `auto_prepend_file` в файле *php.ini*:

```
auto_prepend_file=debug.php
```

Теперь вызовите функцию `pc_debug()` из своей программы, чтобы вывести отладочную информацию:

```
$sql = 'SELECT color, shape, smell FROM vegetables';
pc_debug("[sql: $sql]"); // only printed if DEBUG is true
$r = mysql_query($sql);
```

Обсуждение

Отладочный код – это неизбежное следствие процесса создания программ. Существует ряд приемов, которые помогают быстро локализовать и ликвидировать ошибки. Многие из них включают написание вспомогательного кода, позволяющего убедиться в корректности программы. Чем сложнее программа, тем больше требуется вспомогательного кода. В своей книге «The Mythical Man-Month: Essays on Software Engineering» Фредерик Брукс[1] высказывает предположение, что «вспомогательный код по объему в два раза превосходит код собственно программы». Правильное перспективное планирование позволяет встро-

[1] Брукс Ф. «Мифический человеко-месяц или как создаются программные системы». – Пер. с англ. – СПб: Символ-Плюс, 2000.

ить вспомогательный код в логику программы ясным и эффективным образом. Но для этого требуется заблаговременно обдумать, что именно вы собираетесь измерять и записывать и как вы собираетесь анализировать данные, собранные этим вспомогательным кодом.

Например, можно отсеять информацию, присвоив различные приоритеты различным типам отладочных комментариев. Тогда отладочная функция выведет только те из них, приоритет которых выше текущего.

```
define('DEBUG',2);

function pc_debug($message, $level = 0) {
    if (defined(DEBUG) && ($level > DEBUG) {
        error_log($message);
    }
}

$sql = 'SELECT color, shape, smell FROM vegetables';
pc_debug("[sql: $sql]", 1); // not printed, since 1 < 2
pc_debug("[sql: $sql]", 3); // printed, since 3 > 2
```

Другой прием состоит в разработке функции-оболочки, которая помогает в настройке производительности, предоставляя дополнительную информацию, например время выполнения запросов в базу данных.

```
function getmicrotime(){
    $mtime = microtime();
    $mtime = explode(' ',$mtime);
    return ($mtime[1] + $mtime[0]);
}

function db_query($sql) {
    if (defined(DEBUG) && DEBUG) {
         // начинаем отсчет времени выполнения запроса,
         // если параметр DEBUG установлен в on
         $DEBUG_STRING = "[sql: $sql]<br>\n";
         $starttime = getmicrotime();
    }

    $r = mysql_query($sql);

    if (! $r) {
        $error = mysql_error();
        error_log('[DB: query @'.$_SERVER['REQUEST_URI']."][$sql]: $error");
    } elseif (defined(DEBUG) && DEBUG) {
        // запрос успешный и параметр DEBUG включен,
        // поэтому заканчиваем отсчет времени
        $endtime = getmicrotime();
        $elapsedtime = $endtime - $starttime;
        $DEBUG_STRING .= "[time: $elapsedtime]<br>\n";
        error_log($DEBUG_STRING);
    }

    return $r;
}
```

В данном случае вместо того, чтобы просто занести информацию о выполнении запроса SQL в журнал ошибок, еще записывается количество секунд, которое понадобилось MySQL для выполнения запроса. Это позволяет определить, не выполняется ли какой-нибудь запрос слишком долго.

Функция `getmicrotime()` конвертирует вывод функции `microtime()` в формат, позволяющий без труда выполнять сложение и вычитание чисел.

См. также

Документацию по функции `define()` на *http://www.php.net/define*, по функции `defined()` на *http://www.php.net/defined* и по функции `error_log()` на *http://www.php.net/error-log*; книгу Фредерика П. Брукса (Frederick P. Brooks) «The Mythical Man-Month» (Addison-Wesley).

8.20. Чтение переменных окружения

Задача

Необходимо получить значение переменной окружения.

Решение

Прочитайте значение из суперглобального массива `$_ENV`:

```
$name = $_ENV['USER'];
```

Обсуждение

Переменные окружения – это именованные значения, ассоциированные с процессом. Например, в UNIX можно проверить значение `$_ENV['HOME']` для определения домашнего каталога пользователя:

```
print $_ENV['HOME']; // домашний каталог пользователя
/home/adam
```

Ранние версии PHP автоматически создавали переменные PHP для всех переменных окружения по умолчанию. Начиная с версии 4.1.0 рекомендованный файл *php.ini* запрещает это из соображений скорости выполнения; однако поставляемый файл *php.ini-dist* по-прежнему разрешает загрузку переменных окружения в целях обратной совместимости.

Массив `$_ENV` создается, только если значение параметра конфигурации `variables_order` содержит `E`. Если массив `$_ENV` не разрешен, то для извлечения переменной окружения применяется функция `getenv()`:

```
$path = getenv('PATH');
```

Функция `getenv()` недоступна, если PHP запущен как модуль ISAPI.

См. также

Рецепт 8.21 об установке переменных окружения; документацию по функции `getenv()` на *http://www.php.net/getenv*; информацию о переменных окружения в PHP на *http://www.php.net/reserved.variables.php#reserved.variables.environment*.

8.21. Установка переменных окружения

Задача

Необходимо установить переменную окружения в сценарии или в настройках сервера. Установка переменных окружения в настройках сервера по принципу хост за хостом позволяет конфигурировать виртуальные хосты раздельно.

Решение

Для установки переменной окружения в сценарии применяется функция `putenv()`:

```
putenv('ORACLE_SID=ORACLE'); // конфигурируем расширение oci
```

Функция `SetEnv` позволяет установить переменную окружения в файле Apache *httpd.conf*:

```
SetEnv DATABASE_PASSWORD password
```

Обсуждение

Преимущество определения переменных в файле *httpd.conf* состоит в том, что для них можно установить более строгие ограничения прав доступа на чтение, чем в сценарии PHP. Поскольку процессы веб-сервера должны иметь права на чтение файлов PHP, это в целом дает возможность другим пользователям системы просматривать эти файлы. Сохраняя пароли в файле *httpd.conf*, можно избежать размещения паролей в общедоступном файле. Кроме того, если есть несколько имен хостов, которые ассоциируются с одним и тем же корневым каталогом документов, то можно настроить сценарий так, чтобы его выполнение зависело от имени хоста.

Например, есть хосты *members.example.com* и *guests.example.com*. Версия для хоста members требует аутентификации и предоставляет пользователям дополнительные права доступа. Версия для хоста guests предоставляет ограниченный набор возможностей, но без аутентификации:

```
$version = $_ENV['SITE_VERSION'];

// перенаправляем на http://guest.example.com, если пользователю
// не удалось зарегистрироваться
if ('members' == $version) {
    if (!authenticate_user($_REQUEST['username'], $_REQUEST['password'])) {
```

```
        header('Location: http://guest.example.com/');
        exit;
    }
}

include_once "${version}_header"; // загружаем пользовательский заголовок
```

См. также

Рецепт 8.20 о получении значений переменных окружения; документацию по функции `putenv()` на *http://www.php.net/putenv*; информацию по установке переменных окружения в Apache на *http://httpd.apache.org/docs/mod/mod_env.html*.

8.22. Чтение конфигурационных переменных

Задача

Необходимо получить значение конфигурационной опции PHP.

Решение

Это делается с помощью функции `ini_get()`:

```
// определяем путь включения:
$include_path = ini_get('include_path');
```

Обсуждение

Для получения всех значений переменных конфигурации вызовите функцию `ini_get_all()`. Она возвращает переменные в виде ассоциативного массива, где каждый элемент массива сам является ассоциативным массивом. Второй массив имеет три элемента: глобальное значение для установки, локальное значение и код доступа:

```
// помещаем все переменные конфигурации в ассоциативный массив
$vars = ini_get_all();
print_r($vars['include_path']);
Array
(
    [global_value] => .:/usr/local/lib/php/
    [local_value] => .:/usr/local/lib/php/
    [access] => 7
)
```

Значение `global_value` берется из файла *php.ini*; значение `local_value` принимается во внимание при всех изменениях, сделанных в конфигурационном файле веб-сервера, в любом значимом файле *.htaccess* и в текущем сценарии. Значение `access` — это числовая константа, представляющая место, где это значение можно изменить. Они приведены и прокомментированы в табл. 8.3. Заметим, что имя `access` слегка вво-

дит в заблуждение, поскольку значения параметров всегда могут быть проверены, но их не всегда можно изменить.

Таблица 8.3. Значения параметра access

Значение	Константа PHP	Значение
1	PHP_INI_USER	Любой сценарий, использующий функцию ini_set()
2	PHP_INI_PERDIR	Уровень каталогов, используюший *.htaccess*
4	PHP_INI_SYSTEM	Системный уровень, использующий *php.ini* или *httpd.conf*
7	PHP_INI_ALL	Везде: сценарии, каталоги и система

Если значение равно 6, то установка может быть изменена и на уровне каталогов и на системном уровне (2 + 4 = 6). На самом деле не существует переменных, модифицируемых только на уровне PHP_INI_USER или PHP_INI_PERDIR, но все переменные могут быть изменены на уровне PHP_INI_SYSTEM, поэтому параметр может принимать только значения 4, 6 или 7.

Можно также получить значения параметров, относящихся к определенному расширению, передав имя расширения функции ini_get_all():

```
// возвращаем только переменные, относящиеся к модулю сеанса
$session = ini_get_all('session');
```

По соглашению переменные, относящиеся к определенному расширению, имеют префикс в виде имени расширения и точки. Поэтому, например, все переменные сеанса начинаются с session., а все переменные Java начинаются с java..

Функция ini_get() возвращает текущее значение параметра конфигурации, поэтому для проверки исходного значения из файла *php.ini* применяется функция get_cfg_var():

```
$original = get_cfg_var('sendmail_from'); // мы изменили наш адрес?
```

Функция get_cfg_var() возвращает то же значение, которое появляется в элементе global_value массива, возвращаемого функцией ini_get_all().

См. также

Рецепт 8.23 об установке конфигурационных переменных; документацию по функции ini_get() на *http://www.php.net/ini-get*, по функции ini_get_all() на *http://www.php.net/ini-get-all* и по функции get_cfg_var() на *http://www.php.net/get-cfg-var*; полный список конфигурационных переменных и условия их изменения на *http://www.php.net/function.ini-set.php*.

8.23. Установка конфигурационных переменных

Задача

Необходимо изменить значение параметра конфигурации PHP.

Решение

Это можно сделать с помощью функции `ini_set()`:

```
// добавляем каталог к пути поиска подключаемых файлов
ini_set('include_path', ini_get('include_path') . ':/home/fezzik/php');
```

Обсуждение

Функция `ini_set()` не навсегда изменяет значения переменных конфигурации. Новое значение остается действительным только на время выполнения запроса, в котором вызвана функция `ini_set()`. Чтобы сделать изменения постоянными, измените значения, хранящиеся в файле *php.ini*.

Для некоторых переменных изменение значений не имеет большого смысла, например для `asp_tags` или `register_globals`, поскольку к моменту вызова функции `ini_set()` с целью модификации установок уже слишком поздно менять поведение, на которое эти установки влияют. Если переменная не может быть изменена, то функция `ini_set()` возвращает `false`.

Однако полезно изменять переменные конфигурации на определенных страницах. Например, если вы запускаете сценарий из командной строки, то установите опцию `html_errors` в `off`.

Для восстановления исходного значения переменной применяется функция `ini_restore()`:

```
ini_restore('sendmail_from'); // возвращаем значение по умолчанию
```

См. также

Рецепт 8.22 о получении значений конфигурационных переменных; документацию по функции `ini_set()` на *http://www.php.net/ini-set* и по функции `ini_restore()` на *http://www.php.net/ini-restore*.

8.24. Взаимодействие в рамках Apache

Задача

Необходимо взаимодействовать из PHP с другими частями процесса Apache, обрабатывающего запрос. Это включает установку переменных в файле *access_log*.

Решение

Это делается при помощи функции `apache_note()`:

```
// получаем значение
$session = apache_note('session');

// устанавливаем значение
apache_note('session', $session);
```

Обсуждение

Обрабатывая запрос клиента, Apache совершает ряд шагов, а PHP представляет только одно звено в целой цепи. Apache также переопределяет URL, идентифицирует пользователей, регистрирует запросы и делает многое другое. Во время обработки запроса каждый обработчик имеет доступ к множеству пар ключ/значение, называемому *notes table*. Функция `apache_note()` обеспечивает доступ к этому множеству, чтобы получить информацию, установленную обработчиком на ранней стадии процесса, и оставить эту информацию для обработчика на более поздней стадии.

Например, если для идентификации пользователей и сохранения значений переменных во время запроса используется модуль сеанса, то это можно объединить с анализом файла журнала, и поэтому можно определить среднее количество просмотров страницы в расчете на одного пользователя. Совместное применение функции `apache_note()` и журнального модуля позволяет записать идентификатор сеанса прямо в файл *access_log* для каждого запроса:

```
// извлекаем идентификатор сеанса и добавляем его
// в множество notes table веб-сервера Apache
apache_note('session_id', session_id());
```

Затем модифицируйте файл *httpd.conf*, добавляя следующую строку к `LogFormat`:

```
%{session_id}n
```

Заключительный символ `n` указывает Apache на необходимость использовать переменную, сохраненную в его таблице уведомлений (notes table) другим модулем.

Если PHP скомпилирован с параметром конфигурации `--enable-memory-limit`, то он запоминает максимальный размер памяти, занимаемой каждым запросом, в его записи, называемой `mod_php_memory_usage`. Добавьте информацию о распределении памяти в `LogFormat`:

```
%{mod_php_memory_usage}n
```

См. также

Документацию по функции `apache_note()` на *http://www.php.net/apache-note*; информацию по журналированию в Apache на *http://httpd.apache.org/docs/mod/mod_log_config.html*.

8.25. Профилирование программы

Задача

Есть фрагмент программы, и необходимо провести его исследование, чтобы определить время выполнения каждого оператора.

Решение

Для этого предназначен модуль PEAR Benchmark:

```
require 'Benchmark/Timer.php';

$timer =& new Benchmark_Timer(true);

$timer->start();
// некоторый установочный код
$timer->setMarker('setup');
// еще часть исполняемого кода
$timer->setMarker('middle');
// еще одна часть исполняемого кода
$timer->setmarker('done');
// и наконец последняя часть исполняемого кода
$timer->stop();

$timer->display();
```

Обсуждение

Вызов функции `setMarker()` записывает время. Метод `display()` выводит на печать список маркеров, время их установки и время, прошедшее с момента установки предыдущего маркера:

```
-------------------------------------------------------------
маркер    время установки       прошедшее время    проценты
-------------------------------------------------------------
Start     1029433375.42507400   -                       0.00%
-------------------------------------------------------------
setup     1029433375.42554800   0.00047397613525391    29.77%
-------------------------------------------------------------
middle    1029433375.42568700   0.00013899803161621     8.73%
-------------------------------------------------------------
done      1029433375.42582000   0.00013303756713867     8.36%
-------------------------------------------------------------
Stop      1029433375.42666600   0.00084602832794189    53.14%
-------------------------------------------------------------
total     -                     0.0015920400619507    100.00%
-------------------------------------------------------------
```

Модуль Benchmark также включает класс `Benchmark_Iterate`, позволяющий определить время многократного выполнения одной функции:

```
require 'Benchmark/Iterate.php';

$timer =& new Benchmark_Iterate;
```

```
// простая функция определения времени выполнения
function use_preg($ar) {
    for ($i = 0, $j = count($ar); $i < $j; $i++) {
        if (preg_match('/gouda/',$ar[$i])) {
            // it's gouda
        }
    }
}

// еще одна простая функция определения времени выполнения
function use_equals($ar) {
    for ($i = 0, $j = count($ar); $i < $j; $i++) {
        if ('gouda' == $ar[$i]) {
            // it's gouda
        }
    }
}

// запускаем функцию use_preg() 1000 раз
$timer->run(1000,'use_preg',
                array('gouda','swiss','gruyere','muenster','whiz'));
$results = $timer->get();
print "Mean execution time for use_preg(): $results[mean]\n";

// запускаем функцию use_equals() 1000 раз
$timer->run(1000,'use_equals',
                array('gouda','swiss','gruyere','muenster','whiz'));
$results = $timer->get();
print "Mean execution time for use_equals(): $results[mean]\n";
```

Метод `Benchmark_Iterate::get()` возвращает ассоциативный массив. Элемент `mean` этого массива содержит значение времени выполнения каждой итерации функции. Элемент `iterations` содержит количество итераций. Время выполнения каждой итерации хранится в элементе массива с целочисленным ключом. Например, время первой итерации находится в элементе `$results[1]`, а время 37-й итерации находится в элементе `$results[37]`.

Для автоматической записи времени, прошедшего после выполнения каждой строки программы, применяется конструкция `declare` с параметром `ticks`:

```
function profile($display = false) {
    static $times;

    switch ($display) {
    case false:
        // добавляем текущее время к списку записанных значений времени
        $times[] = microtime();
        break;
    case true:
        // возвращаем прошедшее время в микросекундах
        $start = array_shift($times);
```

```
        $start_mt = explode(' ', $start);
        $start_total = doubleval($start_mt[0]) + $start_mt[1];

        foreach ($times as $stop) {
            $stop_mt = explode(' ', $stop);
            $stop_total = doubleval($stop_mt[0]) + $stop_mt[1];
            $elapsed[] = $stop_total - $start_total;
        }

        unset($times);
        return $elapsed;
        break;
    }
}

// регистрируем обработчик тактов
register_tick_function('profile');

// определяем время старта
profile();

// выполняем код, записывая время выполнения каждого оператора
declare (ticks = 1) {
    foreach ($_SERVER['argv'] as $arg) {
        print strlen($arg);
    }
}

// печатаем прошедшее время
$i = 0;
foreach (profile(true) as $time) {
    $i++;
    print "Line $i: $time\n";
}
```

Параметр `ticks` позволяет многократно выполнять функцию для блока кода. Число `ticks` показывает, сколько тактов должно пройти между вызовами функции, зарегистрированной с помощью `register_tick_function()`.

В предыдущем примере мы регистрируем единственную функцию и выполняем функцию `profile()` для каждого оператора внутри блока `declare`. Если в массиве `$_SERVER['argv']` два элемента, то функция `profile()` выполняется четыре раза: одно выполнение при каждом проходе цикла `foreach` и один раз при каждом выполнении строки `print strlen($arg)`.

Можно также определить вызов двух функций на каждые три оператора:

```
register_tick_function('profile');
register_tick_function('backup');

declare (ticks = 3) {
    // code...
}
```

Можно также передать дополнительные параметры в зарегистрированные функции, которые могут быть методами объекта вместо обычных функций:

```
// передаем "parameter" в функцию profile()
register_tick_function('profile', 'parameter');

// вызываем $car->drive();
$car = new Vehicle;
register_tick_function(array($car, 'drive'));
```

Если необходимо выполнить метод объекта, передайте объект и имя инкапсулированного внутри массива метода. Это даст возможность функции `register_tick_function()` узнать, что вы ссылаетесь на объект, а не на функцию.

Для удаления функции из списка тактовых функций надо вызвать функцию `unregister_tick_function()`:

```
unregister_tick_function('profile');
```

См. также

Информацию о классе PEAR Benchmark на *http://pear.php.net/package-info.php?package=Benchmark*; документацию по функции `register_tick_function()` на *http://www.php.net/register-tick-function*, по функции `unregister_tick_function()` на *http://www.php.net/ unregister-tick-function* и по конструкции *declare* на *http://www.php.net/declare*.

8.26. Программа: (Де)активатор учетной записи на веб-сайте

Не лишним бывает знать, что пользователи, зарегистрировавшиеся на сайте, предоставили корректный адрес электронной почты. Для проверки правильности адреса электронной почты пошлите письмо по адресу, указанному при регистрации. Если пользователь в течение нескольких дней не посетит специальный URL, включенный в письмо, то его учетную запись можно деактивировать.

Система состоит из трех частей. Первая часть, показанная в примере 8.4, представляет собой программу *notify-user.php*, посылающую электронное письмо новому пользователю с просьбой посетить контрольный URL. Вторая часть, приведенная в примере 8.5, – это страница *verify-user.php*, которая обрабатывает контрольный URL и отмечает пользователей, прошедших проверку. Третья часть – это программа *delete-user.php*, деактивирующая учетные записи пользователей, которые не посетили контрольный URL в течение определенного интервала времени. Эта программа показана в примере 8.6.

Ниже приведен оператор SQL, создающий таблицу для хранения информации о пользователе:

```
CREATE TABLE users (
 email VARCHAR(255) NOT NULL,
 created_on DATETIME NOT NULL,
 verify_string VARCHAR(16) NOT NULL,
 verified TINYINT UNSIGNED
);
```

Возможно, вы захотите иметь больше информации о пользователях, но для нашей проверки этого вполне достаточно. При создании учетной записи пользователя занесите информацию в таблицу `users` и пошлите пользователю письмо по электронной почте, объясняющее, как подтвердить его учетную запись. В программе предполагается, что адрес электронной почты пользователя хранится в переменной `$email`.

Пример 8.4. notify-user.php

```
// генерируем контрольную строку
$verify_string = '';
for ($i = 0; $i < 16; $i++) {
    $verify_string .= chr(mt_rand(32,126));
}

// помещаем пользователя в базу данных
if (! mysql_query("INSERT INTO users
(email,created_on,verify_string,verified)
    VALUES ('".addslashes($email)."',NOW(),
            '".addslashes($verify_string)."',0)")) {
    error_log("Can't insert user: ".mysql_error());
    exit;
}

$verify_string = urlencode($verify_string);
$safe_email = urlencode($email);

$verify_url = "http://www.example.com/verify.php";

$mail_body=<<<_MAIL_
To $email:

Please click on the following link to verify your account creation:

$verify_url?email=$safe_email&verify_string=$verify_string

If you do not verify your account in the next seven days, it will be
deleted.
_MAIL_;

mail($email,"User Verification",$mail_body);
```

Контрольная страница, на которую пользователи попадают, если нажимают на ссылку, указанную в почтовом сообщении, обновляет таблицу `users`, если пользователи предоставляют соответствующую информацию, как показано в примере 8.5.

Пример 8.5. verify-user.php

```
$safe_email = addslashes($_REQUEST['email']);
$safe_verify_string = addslashes($_REQUEST['verify_string']);

if ($r = mysql_query("UPDATE users SET verified = 1 WHERE email
    LIKE '$safe_email' AND
    verify_string = '$safe_verify_string' AND verified = 0")) {
    if (mysql_affected_rows() == 1) {
        print "Thank you, your account is verified.";
    } else {
        print "Sorry, you could not be verified.";
    }
} else {
    print "Please try again later due to a database error.";
}
```

Контрольный статус пользователя обновляется, только если адрес электронной почты и контрольная строка, предоставленные пользователем, соответствуют строке в базе данных, которая еще не проверялась. Вот и последний этап – короткая программа, удаляющая непроверенных пользователей по истечении некоторого интервала времени, как показано в примере 8.6.

Пример 8.6. delete-user.php

```
$window = 7; // в днях

if ($r = mysql_query("DELETE FROM users WHERE verified = 0 AND
    created_on < DATE_SUB(NOW(),INTERVAL $window DAY)")) {
    if ($deleted_users = mysql_affected_rows()) {
        print "Deactivated $deleted_users users.\n";
    }
} else {
    print "Can't delete users: ".mysql_error();
}
```

Запускайте эту программу раз в день, чтобы очистить таблицу от пользователей, не прошедших проверку. Если потребуется изменить интервал времени, предоставляемый пользователям для самопроверки, то измените значение `$window` и обновите текст почтового сообщения, посылаемого пользователю, чтобы отразить новое значение.

8.27. Программа: Контролер злоумышленных пользователей

Скорость работы разделяемой памяти делает ее идеальным выбором для хранения данных, к которым необходим частый доступ со стороны различных процессов веб-сервера, когда файлы или база данных работают слишком медленно. В примере 8.7 показан класс `pc_Web_Abuse_Check`, использующий разделяемую память для отслеживания со-

единений с веб-сайтом, для того чтобы отсечь пользователей, которые злоупотребляют веб-сайтом, бомбардируя его запросами.

Пример 8.7. pc_Web_Abuse_Check class

```
class pc_Web_Abuse_Check {
  var $sem_key;
  var $shm_key;
  var $shm_size;
  var $recalc_seconds;
  var $pageview_threshold;
  var $sem;
  var $shm;
  var $data;
  var $exclude;
  var $block_message;

  function pc_Web_Abuse_Check() {
    $this->sem_key = 5000;
    $this->shm_key = 5001;
    $this->shm_size = 16000;
    $this->recalc_seconds = 60;
    $this->pageview_threshold = 30;

    $this->exclude['/ok-to-bombard.html'] = 1;
    $this->block_message =<<<END
<html>
<head><title>403 Forbidden</title></head>
<body>
<h1>Forbidden</h1>
You have been blocked from retrieving pages from this site due to
abusive repetitive activity from your account. If you believe this
is an error, please contact
<a href="mailto:webmaster@example.com?subject=Site+Abuse"
>webmaster@example.com</a>.
</body>
</html>
END;
   }

  function get_lock() {
    $this->sem = sem_get($this->sem_key,1,0600);
    if (sem_acquire($this->sem)) {
      $this->shm = shm_attach($this->shm_key,$this->shm_size,0600);
      $this->data = shm_get_var($this->shm,'data');
    } else {
      error_log("Can't acquire semaphore $this->sem_key");
    }
  }

  function release_lock() {
    if (isset($this->data)) {
      shm_put_var($this->shm,'data',$this->data);
```

```
    }
    shm_detach($this->shm);
    sem_release($this->sem);
  }

  function check_abuse($user) {
    $this->get_lock();
    if ($this->data['abusive_users'][$user]) {
      // если пользователь находится в списке, освобождаем семафор и память
      $this->release_lock();
      //  выводим страницу"you are blocked"
      header('HTTP/1.0 403 Forbidden');
      print $this->block_message;
      return true;
    } else {
     // фиксируем пользователя, находящегося на странице в настоящий момент
     $now = time();
     if (! $this->exclude[$_SERVER['PHP_SELF']]) {
       $this->data['user_traffic'][$user]++;
     }
     // (иногда) идем в начало списка и добавляем плохих людей
     if (! $this->data['traffic_start']) {
       $this->data['traffic_start'] = $now;
     } else {
       if (($now - $this->data['traffic_start']) > $this->recalc_seconds) {
         while (list($k,$v) = each($this->data['user_traffic'])) {
           if ($v > $this->pageview_threshold) {
             $this->data['abusive_users'][$k] = $v;
             // регистрируем добавление пользователя
             // в список злоумышленных пользователей
             error_log("Abuse: [$k] (from ".$_SERVER['REMOTE_ADDR'].')');
           }
         }
         $this->data['traffic_start'] = $now;
         $this->data['user_traffic'] = array();
       }
     }
     $this->release_lock();
    }
    return false;
  }
}
```

Для того чтобы с этим классом можно было работать, в начале страницы вызовите метод `check_abuse()`, передав ему имя зарегистрированного пользователя:

```
// get_logged_in_user_name() - это функция, которая определяет,
// зарегистрировался ли пользователь
if ($user = get_logged_in_user_name()) {
    $abuse = new pc_Web_Abuse_Check();
    if ($abuse->check_abuse($user)) {
```

```
            exit;
        }
    }
```

Метод check_abuse() защищает исключительный доступ к сегменту разделяемой памяти, в котором хранится информация о пользователях и трафике, записанная туда с помощью метода get_lock(). Если пользователь уже находится в списке злоумышленных пользователей, то метод освобождает блок в разделяемой памяти, выдает пользователю страницу ошибки и возвращает значение true. Страница ошибки определяется в конструкторе класса.

Если пользователя нет в списке злоумышленных пользователей и текущая страница (сохраненная в элементе $_SERVER['PHP_SELF']) не входит в список страниц, не подлежащих проверке на злоупотребление, то значение счетчика страниц, просмотренных пользователем, увеличивается. Список страниц, не подлежащих проверке, также определяется в конструкторе. Вызывая метод check_abuse() в начале каждой страницы и помещая страницы, которые не считаются потенциально подверженными злоупотреблению, в массив $exclude, вы тем самым обеспечиваете, что злоумышленный пользователь увидит страницу ошибки даже при просмотре страницы, которая не учитывается при подсчете злоупотреблений. Это делает поведение сайта более уравновешенным.

Следующий раздел функции check_abuse() отвечает за добавление пользователей в список злоумышленных пользователей. Если прошло более $this->recalc_seconds секунд с момента последнего добавления пользователей в список злоумышленных пользователей, то метод смотрит на счетчики посещения страниц каждого пользователя, и если какой-либо из них превышает значение $this-> pageview_threshold, то они добавляются в список злоумышленных пользователей, а в журнал ошибок помещается сообщение. Код, который устанавливает $this->data['traffic_start'], если он еще не установлен, выполняется только при самом первом вызове функции check_abuse(). После добавления нового злоумышленного пользователя функция check_abuse() сбрасывает счетчик пользователей и счетчик просмотра страниц и стартует новый интервал до момента следующего обновления списка злоумышленных пользователей. После освобождения блокировки разделяемой памяти он возвращает false.

Вся информация, необходимая функции check_abuse() для проведения вычислений, т. е. список злоумышленных пользователей, последние значения счетчика просмотренных страниц и время последнего определения злоумышленных пользователей, запоминается в единственном ассоциативном массиве $data. Это делает чтение и запись в разделяемую память более легкой по сравнению с хранением информации в отдельных переменных, поскольку требуется только один вызов функции shm_get_var() и один вызов функции shm_put_var().

Класс pc_Web_Abuse_Check блокирует злоумышленных пользователей, но не имеет никакой системы отчетности и не позволяет добавлять в список или удалять из списка отдельных пользователей. Пример 8.8 содержит программу *abuse-manage.php*, позволяющую манипулировать данными злоумышленных пользователей.

Пример 8.8. abuse-manage.php

```
// класс pc_Web_Abuse_Check определен в abuse-check.php
require 'abuse-check.php';

$abuse = new pc_Web_Abuse_Check();
$now = time();

// обрабатываем команды, если они есть
$abuse->get_lock();
switch ($_REQUEST['cmd']) {
    case 'clear':
      $abuse->data['traffic_start'] = 0;
      $abuse->data['abusive_users'] = array();
      $abuse->data['user_traffic'] = array();
      break;
    case 'add':
      $abuse->data['abusive_users'][$_REQUEST['user']] =
                   'web @ '.strftime('%c',$now);
      break;
    case 'remove':
      $abuse->data['abusive_users'][$_REQUEST['user']] = 0;
      break;
}
$abuse->release_lock();

// теперь значимая информация находится в $abuse->data

print 'It is now <b>'.strftime('%c',$now).'</b><br>';
print 'Current interval started at <b>'.strftime('%c',
$abuse->data['traffic_start']);
print '</b> ('.($now - $abuse->data['traffic_start']).' seconds ago).<p>';

print 'Traffic in the current interval:<br>';
if (count($abuse->data['user_traffic'])) {
  print '<table border="1"><tr><th>User</th><th>Pages</th></tr>';
  while (list($user,$pages) = each($abuse->data['user_traffic'])) {
    print "<tr><td>$user</td><td>$pages</td></tr>";
  }
  print "</table>";
} else {
  print "<i>No traffic.</i>";
}
print '<p>Abusive Users:';

if ($abuse->data['abusive_users']) {
  print '<table border="1"><tr><th>User</th><th>Pages</th></tr>';
  while (list($user,$pages) = each($abuse->data['abusive_users'])) {
```

```
      if (0 === $pages) {
        $pages = 'Removed';
        $remove_command = '';
      } else {
        $remove_command =
           "<a href=\"$_SERVER[PHP_SELF]?cmd=remove&user=".urlencode($user)."\
                    ">remove</a>";
      }
      print "<tr><td>$user</td><td>$pages</td><td>$remove_command</td></tr>";
    }
    print '</table>';
  } else {
    print "<i>No abusive users.</i>";
  }

  print<<<END
  <form method="post" action="$_SERVER[PHP_SELF]">
  <input type="hidden" name="cmd" value="add">
  Add this user to the abusive users list:
  <input type="text" name="user" value="">
  <br>
  <input type="submit" value="Add User">
  </form>
  <hr>
  <form method="post" action="$_SERVER[PHP_SELF]">
  <input type="hidden" name="cmd" value="clear">
  <input type="submit" value="Clear the abusive users list">
  END;
```

Пример 8.8 выводит информацию о текущем значении счетчика количества посещений страницы пользователем и текущий список злоумышленных пользователей, как показано на рис. 8.1. Он также позволяет добавлять в список или удалять из списка определенных пользователей и очищать весь список.

При удалении пользователей из списка вместо вызова:

```
unset($abuse->data['abusive_users'][$_REQUEST['user']])
```

он устанавливает следующую переменную в 0:

```
$abuse->data['abusive_users'][$_REQUEST['user']]
```

Это по-прежнему вынуждает функцию возвращать значение `false`, но позволяет странице точно отметить, что пользователь находился в списке злоумышленных пользователей, хотя и был удален из него. Это полезно в случае, когда удаленный пользователь снова начинает доставлять проблемы.

Когда пользователь добавляется в список злоумышленных пользователей, вместо записи счетчика просмотренных страниц сценарий записывает время, когда пользователь был добавлен. Это полезно, когда

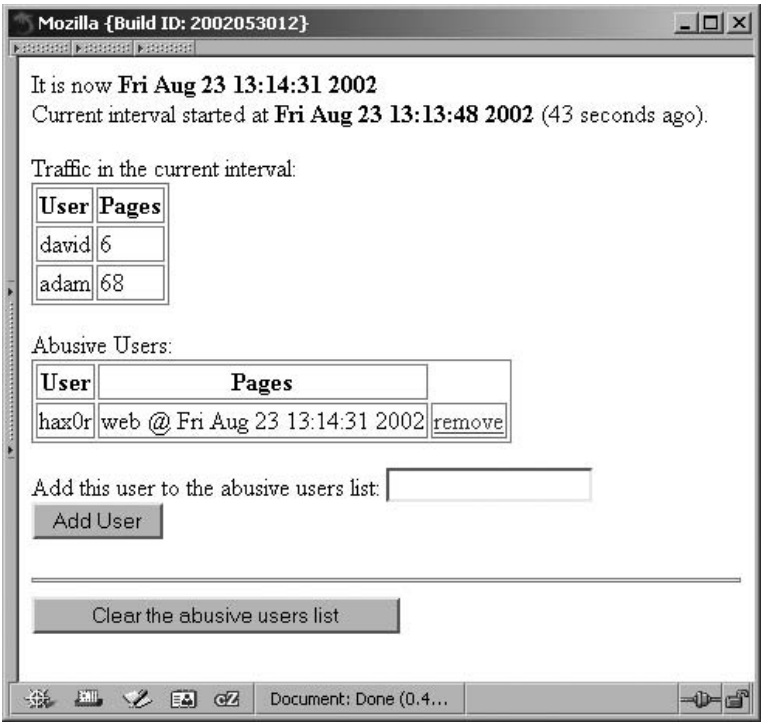

Рис. 8.1. *Злоумышленные пользователи*

нужно выяснить, кто и почему вручную добавил пользователя в список злоумышленных пользователей.

Если вы помещаете `pc_Web_Abuse_Check` и эту страницу поддержки на вашем сервере, то обеспечьте защиту страницы поддержки с помощью пароля или каким-то другим способом от публичного доступа. Очевидно, что от этого кода не очень много толку, если злоумышленные пользователи могут удалить себя из списка злоумышленных пользователей.

9

Формы

9.0. Введение

Гениальность PHP – в той простоте, с которой он позволяет интегрировать в программу переменные форм. Это делает веб-программирование приятным и простым – от вывода формы и ее обработки до вывода результатов.

В HTTP нет встроенного механизма, который позволял бы сохранять информацию при переходе от одной страницы к другой. Причина этого в том, что HTTP не сохраняет свое состояние между запросами. Рецепты 9.1, 9.3, 9.4 и 9.5 показывают, как обойти фундаментальную проблему определения, какой пользователь посылает запрос на веб-сервер.

Обработка информации, получаемой от пользователя, – это еще одна основная тема этой главы. Вы никогда не должны доверять данным, доставленным броузером, поэтому обязательной является проверка достоверности всех полей, даже скрытых элементов формы. Существует много способов проверки корректности данных – от проверки информации на соответствие определенному критерию, что обсуждается в рецепте 9.2, до преобразования сущностей HTML в escape-последовательности, что позволяет безопасно отображать информацию, вводимую пользователем, как показано в рецепте 9.8. Кроме того, в рецепте 9.7 рассказано, как обеспечить безопасность веб-сервера, а в рецепте 9.6 рассмотрена обработка файлов, загружаемых пользователем.

При обработке страницы PHP всегда устанавливает наличие переменных, пришедших с запросами GET или POST, загруженных файлов, допустимых cookies, а также переменных веб-сервера и окружения. После этого все эти данные становятся доступными посредством обращения к следующим массивам: `$_GET`, `$_POST`, `$_FILES`, `$_COOKIE`, `$_SERVER` и `$_ENV`. Они содержат, соответственно, все переменные, установленные запросами GET, запросами POST, загруженными файлами, cookies, веб-сервером и окружением. Кроме того, есть массив `$_REQUEST`, содержащий пришедшие от пользователя данные – GET, POST и cookies.

Если два массива содержат ключ с одним и тем же именем, то при размещении элементов внутри массива $_REQUEST PHP поступает в соответствии с параметром конфигурации variables_order. По умолчанию variables_order равна EGPCS (или GPCS, если используется файл конфигурации *php.ini-recommended*). Поэтому PHP сначала добавляет в массив $_REQUEST переменные окружения, а затем добавляет переменные GET, POST, cookie и переменные веб-сервера в указанном порядке. Например, если по умолчанию C идет после P, то cookie с именем username переписывает переменную POST с именем username.

Если у вас нет доступа к файлу конфигурации PHP, то проверить установку переменной можно с помощью функции ini_get():

```
print ini_get('variables_order');
EGPCS
```

Вам может потребоваться сделать это, поскольку ваш интернет-провайдер запрещает вам просматривать параметры конфигурации, или потому, что ваш сценарий, вероятно, запущен еще на каком-нибудь сервере. Эти параметры можно также просмотреть посредством функции phpinfo(). Однако если нельзя полагаться на значение опции variables_order, то надо обратиться непосредственно к массивам $_GET и $_POST, вместо того чтобы использовать массив $_REQUEST.

Массивы, содержащие внешние переменные, такие как $_REQUEST, являются суперглобальными. Это означает, что их не требуется объявлять как global внутри функции или класса. Это также означает, что, вероятно, вы не должны присваивать что-нибудь этим переменным, иначе вы перепишете хранимую в них информацию.

До версии PHP 4.1 эти суперглобальные переменные не существовали. Вместо них были обычные массивы с именами $HTTP_COOKIE_VARS, $HTTP_ENV_VARS, $HTTP_GET_VARS, $HTTP_POST_VARS, $HTTP_POST_FILES и $HTTP_SERVER_VARS. Эти массивы все еще доступны в целях совместимости, но с новыми массивами работать проще. Эти более старые массивы заполняются, только если параметр конфигурации track_vars установлен в on, но начиная с версии PHP 4.0.3 эта возможность включена всегда.

В заключение, если параметр конфигурации register_globals установлен в on, все эти переменные также доступны как переменные глобального пространства имен. Поэтому $_GET['password'] — это одновременно и просто $password. Удобству при этом сопутствуют значительные проблемы безопасности, поскольку злоумышленные пользователи могут легко установить переменные извне и переписать внутренние переменные, которым вы, вроде бы, должны доверять. Начиная с версии PHP 4.2 параметр register_globals по умолчанию устанавливается в off.

Опираясь на полученные знания, напишем сценарий, объединяющий все сказанное выше. Форма просит пользователя ввести его имя, а затем выводит сообщение с приветствием. HTML-документ для этой формы может выглядеть следующим образом:

```
<form action="/hello.php" method="post">
Как Ваше имя?
<input type="text" name="first_name">
<input type="submit" value="Поздоровайтесь">
</form>
```

Параметр `name` текстового элемента `input` внутри формы имеет значение `first_name`. Кроме того, в форме используется метод `post`. Это значит, что после отправки формы элемент `$_POST['first_name']` будет содержать любую строку, которую напечатает пользователь. (Она может быть также пустой, если, конечно, ничего не было напечатано.)

Но давайте для простоты предположим, что в переменной находится допустимое значение. («Допустимое» может означать, в зависимости от обстоятельств, «непустое», «не заданное в попытке взломать систему» и т. д.) Это позволит нам пропустить важный этап проверки ошибок, но зато мы сможем представить этот простой пример. Итак, ниже показан простой сценарий *hello.php* для обработки формы:

```
echo 'Hello ' . $_POST['first_name'] . '!';
```

Если пользователя зовут Joe, то PHP печатает:

```
Hello Joe!
```

9.1. Обработка информации, полученной из формы

Задача

Необходимо использовать одну и ту же страницу HTML для вывода формы и обработки введенных в ней данных. Другими словами, требуется избежать размножения страниц, которые работают на отдельных этапах транзакции.

Решение

Используйте скрытое поле в форме, чтобы указать программе, что предполагается его обработка в форме. В данном случае скрытое поле называется `stage` и имеет значение `process`:

```
if (isset($_POST['stage']) && ('process' == $_POST['stage'])) {
    process_form();
} else {
    print_form();
}
```

Обсуждение

Когда люди создавали формы на заре развития Всемирной паутины, они делали две страницы: статическую HTML-страницу с формой и

сценарий, который обрабатывал форму и возвращал динамически сгенерированный ответ пользователю. Это было немного громоздко, поскольку *form.html* была источником для *form.cgi*, и если одна страница изменялась, то нужно было не забыть также отредактировать и другую, иначе сценарий мог работать неправильно.

Формы легче поддерживать, когда все части находятся в том же самом файле, а контекст определяет, какие разделы отображать. Используйте скрытое поле формы с именем `stage`, чтобы отслеживать позицию в процессе обработки формы; оно действует как диспетчер этапов, возвращающих пользователю соответствующий HTML-документ. Однако иногда такой подход невозможен; например, когда ваша форма обрабатывается сценарием на каком-нибудь другом сервере.

Однако, создавая HTML-документ для формы, не прописывайте жестко путь к странице в атрибуте `action`. Это делает невозможным переименование и изменение местоположения страницы без одновременного ее редактирования. Вместо этого PHP предоставляет полезную переменную:

```
$_SERVER['PHP_SELF']
```

Эта переменная является синонимом URL текущей страницы. Поэтому установите атрибут `action` в это значение, и ваша форма всегда будет отправляться, даже если вы переместили файл в новое место на сервере.

Поэтому пример во введении этой главы теперь выглядит следующим образом:

```
if (isset($_POST['stage']) && ('process' == $_POST['stage'])) {
    process_form();
} else {
    print_form();
}

function print_form() {
    echo <<<END
        <form action="$_SERVER[PHP_SELF]" method="post">
        What is your first name?
        <input type="text" name="first_name">
        <input type="hidden" name="stage" value="process">
        <input type="submit" value="Say Hello">
        </form>
END;
}

function process_form() {
    echo 'Hello ' . $_POST['first_name'] . '!';
}
```

Если форма имеет более одного этапа, то просто устанавливайте атрибут `stage` в новое значение для каждого этапа.

См. также

Рецепт 9.3 об обработке многостраничных форм.

9.2. Проверка корректности введенных в форму данных

Задача

Необходимо гарантировать, что данные, введенные в форму, удовлетворяют определенному критерию.

Решение

Создайте функцию, которая принимает строку для проверки и возвращает `true`, если строка прошла проверку, и `false`, если не прошла. Внутри функции используйте регулярные выражения и сравнения для проверки данных. Так, пример 9.1 показывает функцию `pc_validate_zipcode()`, которая проверяет достоверность почтового индекса США.

Пример 9.1. pc_validate_zipcode( )

```
Function pc_validate_zipcode($zipcode) {
    Return preg_match('/^[0-9]{5}([- ]?[0-9]{4})?$/', $zipcode);
}
```

Ниже показано, как ее использовать:

```
if (pc_validate_zipcode($_REQUEST['zipcode'])) {
    // почтовый индекс США корректный, можно продолжать
    process_data();
} else {
    // это неправильный почтовый индекс, печатаем сообщение об ошибке
    print "Your ZIP Code is should be 5 digits (or 9 digits, if you're ";
    print "using ZIP+4).";
    print_form();
}
```

Обсуждение

Какие данные назвать корректными, а какие некорректными, вопрос скорее философский, и ответ на него трудно облечь в конкретную форму последовательной серии фиксированных шагов. Во многих случаях то, что может абсолютно подходить в одной ситуации, может быть неверным в другой.

Легче всего проверить, что поле не пустое. Эту задачу наилучшим образом решает функция `empty()`.

Далее идут относительно простые проверки, такие как в случае с почтовым индексом США. Обычно одно или два регулярных выражения помогают справиться с этой проблемой. Например:

```
/^[0-9]{5}([- ]?[0-9]{4})?$/
```

определяет все корректные почтовые индексы США.

Однако иногда обеспечить соответствие корректным регулярным выражениям трудно. Если нужно проверить, что введенное имя состоит из двух слов, например «Alfred Aho», можно провести сравнение с:

```
/^[A-Za-z]+ +[A-Za-z]+$/
```

Однако Tim O'Reilly не сможет пройти проверку. Альтернативой является `/^\S+\s+\S+$/`; но тогда отвергается Donald E. Knuth. Поэтому перед составлением регулярных выражений следует тщательно продумывать весь диапазон допустимых входных данных.

В некоторых случаях даже при помощи регулярных выражений бывает трудно проверить, является ли значение поля допустимым. Чрезвычайно популярной и сложной является задача проверки на подлинность адресов электронной почты, рассматриваемая в рецепте 13.6. Другая задача – это убедиться, что пользователь ввел правильное название своего американского штата. Можно проверить путем сравнения со списком названий, но что если он ввел сокращение своей почтовой службы? Сработает ли MA вместо Massachusetts? А как насчет Mass.?

Один из способов обойти эту трудность состоит в том, чтобы предоставить пользователю выпадающий список заранее сгенерированных вариантов. Применение элемента `select` в конструкции формы заставляет пользователя выбрать штат в формате, который всегда работает, что может уменьшить количество ошибок. Однако это приводит еще к ряду затруднений. Что если пользователь живет в месте, которое не попадает в список вариантов? Что если диапазон вариантов настолько велик, что является неосуществимым решением?

Существует множество путей решения такого типа проблем. Во-первых, можно добавить в список пункт «other», чтобы пользователь не из США мог успешно заполнить форму. (В противном случае он может выбрать место произвольно – просто чтобы продолжать пользоваться вашим сайтом.) Во-вторых, можно разбить процесс регистрации на две последовательные части. В случае длинного списка опций пользователь вначале выбирает букву алфавита, на которую начинается его вариант; затем новая страница предоставляет ему список, содержащий только варианты, начинающиеся на эту букву.

Наконец, есть и более сложные проблемы. Что делать, если необходимо удостовериться, что пользователь ввел корректную информацию, но нельзя говорить ему об этом? Это важно в лотереях; в лотереях часто применяется специальное кодовое окно во входной форме, в котором пользователь вводит строку `AD78DQ` из электронной почты или из

рекламы, которую он принял. Вы хотите быть уверены, что здесь нет опечатки, в противном случае ваша программа не будет рассматривать его как законного посетителя. Нежелательно также позволить пользователю просто угадать коды, поскольку он может попытаться подобрать их и взломать систему.

Решение состоит в том, чтобы иметь два окна ввода. Пользователь вводит свой код дважды; если два поля совпадают, то данные считаются легальными, и затем (молча) проверяется их достоверность. Если значения полей не совпадают, то ввод отвергается, и пользователя просят его исправить. Эта процедура исключает опечатки и не раскрывает, как работает алгоритм проверки достоверности данных; она также предотвращает появление орфографических ошибок в адресах электронной почты.

В заключение надо заметить, что PHP выполняет проверку достоверности данных на стороне сервера. Такая проверка требует, чтобы был сделан запрос на сервер и в ответ получена страница, что может занимать длительное время. Можно также выполнять проверку корректности данных на стороне клиента с помощью JavaScript. Хотя проверка на стороне клиента и работает быстрее, но при этом код проверки становится виден пользователю, а также он может не работать, если клиент не поддерживает или отключил JavaScript. Поэтому необходимо всегда дублировать на сервере все программы проверки на стороне клиента.

См. также

Рецепт 13.6, где рассказывается о применении регулярных выражений для проверки достоверности адресов электронной почты; главу 7 «Validation on the Server and Client» в книге «*Web Database Applications with PHP and MySQL*» Хьюга Вильямса (Hugh Williams) и Дэвида Лэйна (David Lane) (O'Reilly).

9.3. Работа с многостраничными формами

Задача

Необходимо использовать форму, которая показывает более одной страницы и сохраняет данные при переходе от одной страницы к следующей.

Решение

Используйте сеансы:

```
session_start();
$_SESSION['username'] = $_GET['username'];
```

Можно также включать переменные из формы более ранней страницы в качестве скрытых полей ввода в более поздних страницах:

```
<input type="hidden" name="username"
       value="<?php echo htmlentities($_GET['username']); ?>">
```

Обсуждение

При любой возможности используйте сеансы. Это более безопасно, поскольку пользователь не может модифицировать переменные сеанса. Чтобы начать сеанс, вызовите функцию `session_start()`, которая создаст новый сеанс или продолжит существующий. Заметим, что этот этап не является необходимым, если в файле *php.ini* установлен параметр `session.auto_start`. Переменные, присвоенные массиву `$_SESSION`, автоматически передаются между сценариями. В примере, показанном в разделе «Решение», переменная формы `username` сохраняется с помощью присваивания элемента `$_GET['username']` элементу `$_SESSION['username']`.

Для получения доступа к этому значению в последующем запросе вызовите функцию `session_start()`, а затем проверьте элемент `$_SESSION['username']`:

```
session_start();
$username = htmlentities($_SESSION['username']);
print "Hello $username.";
```

В данном случае, если не вызвать функцию `session_start()`, то массив `$_SESSION` не будет установлен.

Обеспечьте безопасность сервера и сохраните в секрете место расположения файлов сеанса (системных файлов, базы данных и т. д.); в противном случае ваша система будет уязвима для соединений с ложной аутентификацией.

Если использование сеансов не разрешено для вашей инсталляции PHP, то взамен можно использовать скрытые переменные формы. Однако передача данных с помощью скрытых элементов формы небезопасна, поскольку кто-нибудь может отредактировать эти поля и подделать запрос; но, приложив небольшие усилия, можно повысить безопасность до приемлемого уровня.

Основной способ применения скрытых полей состоит в том, чтобы включить их внутрь формы.

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
      method="get">

<input type="hidden" name="username"
       value="<?php echo htmlentities($_GET['username']); ?>">
```

После того как эта форма будет отправлена повторно, элемент `$_GET['username']` будет содержать предыдущее значение, если только кто-нибудь не поменяет его.

Есть и более сложное, но безопасное решение – преобразовать переменные в строку с помощью функции `serialize()`, вычислить секретный хеш из данных и поместить обе части информации в форму. Затем в следующем запросе проверьте достоверность данных и выполните обратное преобразование. Если данные не пройдут проверки на достоверность, вы будете знать, что кто-то пытался модифицировать информацию.

Кодирующая функция `pc_encode()`, показанная в примере 9.2, принимает данные для декодирования в виде массива.

Пример 9.2. pc_encode( )

```
$secret = 'Foo25bAr52baZ';

function pc_encode($data) {
  $data = serialize($data);
  $hash = md5($GLOBALS['secret'] . $data);
  return array($data, $hash);
}
```

В функции `pc_encode()` данные преобразуются в строку, вычисляется контрольный хеш и эти переменные возвращаются.

Функция `pc_decode()`, показанная в примере 9.3, делает работу обратную той, которую выполнил ее двойник.

Пример 9.3. pc_decode( )

```
function pc_decode($data, $hash) {
  if (!empty($data) && !empty($hash)) {
    if (md5($GLOBALS['secret'] . $data) == $hash) {
      return unserialize($data);
    } else {
      error_log("Validation Error: Data has been modified");
      return false;
    }
  }
  return false;
}
```

Функция `pc_decode()` вновь создает хеш секретного слова с данными и сравнивает его со значением хеша из формы. Если они равны, то переменная `$data` считается достоверной и поэтому над ней выполняется обратное преобразование. Если проверка заканчивается неудачей, то функция записывает сообщение в журнал ошибок и возвращает `false`.

Эти функции работают вместе следующим образом:

```
<?php
$secret = 'Foo25bAr52baZ';

// Загружаем старые данные и проверяем их достоверность
if (! $data = pc_decode($_GET['data'], $_GET['hash'])) {
  // попытка взлома
}
```

```
// Обрабатываем форму (новые данные формы находятся в $_GET)

// Обновляем $data
$data['username'] = $_GET['username'];
$data['stage']++;
unset($data['password']);

// Кодируем результаты
list ($data, $hash) = pc_encode($data);

// Сохраняем данные и хеш внутри формы
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="get">
...

<input type="hidden" name="data"
       value="<?php echo htmlentities($data); ?>">
<input type="hidden" name="hash"
       value="<?php echo htmlentities($hash); ?>">
</form>
```

В начале сценария мы передаем функции `pc_decode()` переменные из формы для декодирования. Как только информация загружена в массив `$data`, обработку формы можно продолжить, проверяя новые переменные в массиве `$_GET`, а старые переменные в `$data`. После завершения проверки обновляем массив `$data`, сохраняя в нем новые переменные, вычисляя по пути хеш. Наконец, выводим новую форму и включаем `$data` и `$hash` как скрытые переменные.

См. также

Информацию об использовании модуля сеанса в рецептах 8.5 и 8.6; рецепт 9.8, где детально описывается использование функции `htmlentities()` для преобразования в escape-последовательности управляющих символов в выводе HTML; рецепт 14.3 о проверке данных с помощью хеша; документацию по слежению за сеансом на *http://www.php.net/session* и в рецепте 8.4; документацию по функции `serialize()` на *http://www.php.net/serialize* и по функции `unserialize()` на *http://www.php.net/unserialize*.

9.4. Повторный вывод форм с информацией и сообщениями об ошибках

Задача

Когда возникает проблема с данными, введенными в форму, необходимо напечатать сообщения об ошибках рядом с проблемными полями, вместо генерации сообщения об ошибке в начале формы. Также требуется сохранить значения, которые пользователь напечатал в форме сначала.

Решение

Сохраните сообщения в массиве `$errors`, индексируя их по именам полей.

```
if (! pc_validate_zipcode($_REQUEST['zipcode'])) {
    $errors['zipcode'] = "This is is a bad ZIP Code. ZIP Codes must "
                       . "have 5 numbers and no letters.";
}
```

При повторном выводе формы можно показывать каждую ошибку с помощью поля, помещая в него исходное значение:

```
echo $errors['zipcode'];
$value = isset($_REQUEST['zipcode']) ?
               htmlentities($_REQUEST['zipcode']) : '';
echo "<input type=\"text\" name=\"zipcode\" value=\"$value\">";
```

Обсуждение

Если пользователи сталкиваются с ошибками при заполнении длинной формы, то можно повысить общее удобство и простоту использования формы, четко выделяя место, где нужно исправить ошибки.

Объединение всех ошибок в одном массиве дает много преимуществ. Прежде всего, нетрудно определить, нашлись ли в результате проверки на достоверность информации какие-либо элементы, требующие исправления; просто используйте функцию `count($errors)`. Использовать такой метод проще, чем следить за этим событием с помощью отдельной переменной, особенно, если поток сложный или реализуется с помощью множества функций. В примере 9.4 показана функция проверки на достоверность `pc_validate_form()`, которая использует массив `$errors`.

Пример 9.4. pc_validate_form( )

```
function pc_validate_form() {
  if (! pc_validate_zipcode($_POST['zipcode'])) {
     $errors['zipcode'] = "ZIP Codes are 5 numbers";
  }

  if (! pc_validate_email($_POST['email'])) {
     $errors['email'] = "Email addresses look like user@example.com";
  }

  return $errors;
}
```

Это ясный код, поскольку все ошибки сохраняются в одной переменной. Переменную можно легко передать куда угодно, если вы не хотите, чтобы она находилась в глобальной области видимости.

Использование имени переменной в качестве ключа сохраняет связи между полем, которое явилось причиной ошибки, и самим сообщени-

ем об ошибке. Эти связи также облегчают выполнение цикла по элементам при показе ошибок.

Можно автоматизировать скучную задачу вывода формы. Функция pc_print_form() в примере 9.5 показывает, как это сделать.

Пример 9.5. pc_print_form( )

```
function pc_print_form($errors) {
    $fields = array('name'   => 'Name',
                    'rank'   => 'Rank',
                    'serial' => 'Serial');

    if (count($errors)) {
        echo 'Please correct the errors in the form below.';
    }

    echo '<table>';

    // выводим ошибки и переменные формы
    foreach ($fields as $field => $field_name) {
        // открываем строку
        echo '<tr><td>';

        // печатаем ошибку
        if (!empty($errors[$field])) {
            echo $errors[$field];
        } else {
            echo ' '; // чтобы не портить внешний вид таблиц
        }

        echo "</td><td>";

        // печатаем имя и введенную информацию
        $value = isset($_REQUEST[$field]) ?
                        htmlentities($_REQUEST[$field]) : '';

        echo "$field_name: ";
        echo "<input type=\"text\" name=\"$field\" value=\"$value\">";
        echo '</td></tr>';
    }

    echo '</table>';
}
```

Сложная часть функции pc_print_form() начинается с массива $fields. Ключ – это имя переменной; значением является подходящее для показа имя поля. Определив их в начале функции, можно создать цикл по значениям с помощью оператора foreach; в противном случае понадобятся три отдельные строки с идентичным кодом. К этому добавляется использование имени переменной в качестве ключа в массиве $errors, поскольку можно найти сообщение об ошибке внутри цикла, просто проверяя элемент $errors[$field].

Если необходимо распространить этот пример на поля ввода, отличные от `text`, модифицируйте массив `$fields`, включив дополнительную метаинформацию о полях формы:

```
$fields = array('name' => array('name' => 'Name', 'type' => 'text'),
                'rank' => array('name' => 'Rank', 'type' => 'password'),
                'serial' => array('name' => 'Serial', 'type' => 'hidden')
               );
```

См. также

Рецепт 9.2, где показан простой способ проверки достоверности формы.

9.5. Защита от многократной отправки одной и той же формы

Задача

Необходимо помешать пользователям отправлять одну и ту же форму несколько раз.

Решение

Сгенерируйте уникальный идентификатор и сохраните эту метку в скрытом поле формы. Перед обработкой формы проверьте, не была ли эта метка уже представлена. Если нет, то можно продолжать, а если да, то надо сгенерировать ошибку.

Для создания формы применяется функция `uniqid()`, чтобы получить уникальный идентификатор:

```
<?php
$unique_id = uniqid(microtime(),1);
...
?>
<input type="hidden" name="unique_id" value="<?php echo $unique_id; ?>">
</form>
```

Затем в процессе обработки ищите этот идентификатор:

```
$unique_id  = $dbh->quote($_GET['unique_id']);
$sth = $dbh->query("SELECT * FROM database WHERE unique_id = $unique_id");

if ($sth->numRows()) {
    // уже была представлена, выдаем ошибку
} else {
   // работаем с данными
}
```

Обсуждение

Пользователи повторно отправляют форму по множеству причин. Часто это всего лишь ошибочный щелчок по кнопке мыши – двойной вместо одинарного. Пользователь может нажать кнопку Back своего броузера, чтобы отредактировать или повторно проверить информацию, а затем снова нажать кнопку Submit вместо кнопки Forward. Это может быть сделано умышленно: он пытается повторно проголосовать в онлайновом опросе или лотерее. Код, представленный в разделе «Решение», предохраняет от непредумышленных атак и замедляет работу злоумышленных пользователей. Однако он не исключает все варианты жульнического использования: для этого требуется проделать более сложную работу.

Решение предохраняет базу данных от переполнения слишком большим количеством копий одной и той же записи. С помощью генерации метки, размещаемой в форме, можно однозначно идентифицировать этот конкретный экземпляр формы, даже если cookies запрещены. Записывая впоследствии данные из формы, вы сохраняете вместе с ними и эту метку. Это позволяет легко определить, не видели ли вы уже эту форму и запись в базе данных, связанную с ней.

Начните с добавления в таблицу базы данных дополнительной колонки unique_id, предназначенной для хранения идентификатора. При добавлении данных в запись вставьте также и идентификатор. Например:

```
$username  = $dbh->quote($_GET['username']);
$unique_id = $dbh->quote($_GET['unique_id']);

$sth = $dbh->query("INSERT INTO members ( username,  unique_id)
                                 VALUES ($username, $unique_id)");
```

Ассоциируя правильную строку в базе данных с формой, можно легко обрабатывать повторную отправку. Однозначного решения не существует, оно зависит от ситуации. В некоторых случаях нужно одновременно игнорировать второе представление. В других случаях необходимо проверить, изменялась ли запись, и если да, то выдать пользователю диалоговое окно с вопросом: хочет ли он обновить запись или оставить старые данные. Наконец, чтобы повторно отобразить форму подтверждения, можно молча обновить запись, и пользователь никогда не узнает о проблеме.

Надо иметь в виду, что все эти возможности обладают различными особенностями взаимодействия с пользователем. По нашему мнению, нет никаких причин позволять недостаткам HTTP определять квалификацию пользователей. Поэтому, хотя третий вариант – обновление записи без предупреждения – и не является нормальным поведением, он во многих отношениях представляет наиболее естественный выбор. Приложения, которые мы разработали, применяя этот метод, наиболее дружественны к пользователю; другие два способа запутывают или разочаровывают большинство пользователей.

Заманчиво избежать генерации случайной метки и вместо нее использовать число, на единицу превышающее количество записей, уже находящихся в базе данных. Таким образом, метка и первичный ключ будут совпадать, и не потребуется использовать дополнительную колонку. Есть две (по крайней мере) проблемы с этим методом. Во-первых, он создает условия гонок. Что происходит, когда второй человек запускает форму до того, как первый человек закончит с ней работу? Вторая форма будет тогда иметь ту же самую метку, что и первая, и возникнет конфликт. Это можно обойти, создавая новую, пустую запись в базе данных при запросе формы, поэтому второй человек получит число на единицу большее, чем первый. Однако это приведет к появлению пустых строк в базе данных, если пользователи предпочтут не заполнять форму.

Другая причина, по которой не следует так делать, состоит в том, что при этом можно легко отредактировать другую запись в базе данных, вручную настроив идентификатор на другое число. В зависимости от настроек безопасности, ложные представления GET или POST позволяют без проблем изменить данные. Тем не менее длинную случайную метку нельзя угадать простой заменой на другое целое число.

См. также

Подробности о проверке данных с помощью хеша в рецепте 14.3; документацию по функции `uniqid()` на *http://www.php.net/uniqid*.

9.6. Обработка загруженных файлов

Задача

Необходимо обработать файл, загруженный пользователем.

Решение

Используйте массив `$_FILES`:

```
// из <input name="event" type="file">
if (is_uploaded_file($_FILES['event']['tmp_name'])) {
    readfile($_FILES['event']['tmp_name']); // выводим файл на экран
}
```

Обсуждение

Начиная с версии PHP 4.1 все загруженные файлы появляются в суперглобальном массиве `$_FILES`. Для каждого файла в нем есть четыре информационных раздела:

`name`

Имя, присвоенное элементу ввода формы.

type

Тип MIME-файла.

size

Размер файла в байтах.

tmp_name

Временное местоположение файла на сервере.

В более ранних версиях PHP вместо этого массива необходимо использовать массив $HTTP_POST_FILES.[1]

После выбора файла из массива используйте функцию is_uploaded_file() для подтверждения того, что файл, который нужно обработать, действительно загружен пользователем, затем обработайте его таким же образом, как и другие файлы в системе. Всегда поступайте таким образом. Если вы слепо доверяете именам файлов, предоставленных пользователем, кто-нибудь может изменить запрос и добавить имена, такие как */etc/passwd* в список для обработки.

Можно также переместить файл на постоянное местоположение; для безопасного перемещения файла используйте функцию move_uploaded_file():

```
// перемещаем файл: функция move_uploaded_file() также выполняет
// проверку файлов на легитимность, поэтому нет необходимости
// вызывать еще и функцию is_uploaded_file()
move_uploaded_file($_FILES['event']['tmp_name'], '/path/to/file.txt');
```

Обратите внимание, что значение, сохраненное в переменной tmp_name, представляет собою полный путь к файлу, а не просто имя файла. Используйте функцию basename(), чтобы выделить имя, если это необходимо.

Не забудьте убедиться, что PHP имеет права на чтение и запись и в каталоге, куда записываются временные файлы (см. параметр конфигурации upload_tmp_dir для проверки его местоположения) и в каталоге, куда вы собираетесь копировать файл. Часто это может быть пользователь nobody или apache (вместо вашего персонального имени пользователя). Вследствие этого, если вы работаете в режиме safe_mode, после копирования файла в новый каталог, возможно, у вас уже не будет к нему доступа.

Обработка файлов нередко представляет собой нетривиальную задачу, поскольку не все броузеры одинаково представляют одну и ту же информацию. Делать это следует делать очень аккуратно, в противном случае вы можете сами создать брешь в безопасности. В конце концов, злоумышленники могут получить возможность загружать на вашу машину любые файлы, взломать или разрушить вашу систему.

---

[1] Начиная с версии 4.2 в массив добавлен пятый раздел – *error*. Более подробную информацию можно найти по адресу *http://www.php.net/manual/features.file-upload.php*. – *Примеч. науч. ред.*

Поэтому в PHP реализовано несколько возможностей устанавливать ограничения на загружаемые файлы, включая полный запрет на загрузку любых файлов. Поэтому, если вы испытываете трудности при обработке загруженных файлов, убедитесь, что файл не отвергается из-за того, что он может представлять риск для безопасности.

Чтобы выполнить такую проверку, сначала убедитесь, что параметр `file_uploads` в файле конфигурации установлен в `on`. Затем проверьте, что размер файла не превышает значение `upload_max_filesize`; по умолчанию оно равно 2 Мбайт, что блокирует чьи-либо попытки разрушить систему путем заполнения жесткого диска гигантскими файлами. Кроме того, есть параметр `post_max_size`, контролирующий максимальный размер всех данных POST, разрешенный в одном запросе; начальное значение равно 8 Мбайт.

Вследствие возможных различий в броузерах и ошибок пользователей, если вы не можете получить массив `$_FILES` для заполнения его информацией, убедитесь, что вы добавили атрибут `enctype="multipart/form-data"` в открывающий тег формы – это необходимо PHP для запуска обработки. Если вы не можете это сделать, то придется вручную анализировать `$HTTP_RAW_POST_DATA`. (См. спецификацию MIME в RFC 1521 и 1522 на *http://www.faqs.org/rfcs/rfc1521.html* и *http://www.faqs.org/rfcs/ rfc1522.html*.)

Кроме того, если не выбрано ни одного файла для загрузки, то версии PHP до 4.1 устанавливают параметр `tmp_name` в `none`; более новые версии присваивают ему пустую строку. Версия PHP 4.2.1 допускает файлы нулевой длины. Чтобы убедиться, что файл загружен и он не пустой (хотя, в зависимости от обстоятельств, это как раз то, что нужно), необходимо убедиться, что параметр `tmp_name` установлен, а значение параметра `size` больше нуля. И наконец, не все броузеры обязательно посылают тот же тип MIME-файла; то, что они посылают, зависит от того, какие типы файлов они допускают.

См. также

Документацию по обработке загруженных файлов на *http://www.php.net/features.file-upload* и по функции `basename()` на *http://www.php.net/basename*.

9.7. Организация безопасности обработки форм в PHP

Задача

Необходимо обеспечить безопасную обработку входных переменных формы и не позволить какому-нибудь злоумышленнику изменить переменные в вашей программе.

Решение

Заблокируйте параметр конфигурации register_globals и обращайтесь только к переменным из массива $_REQUEST. Чтобы обеспечить еще более крепкую защиту, используйте массивы $_GET, $_POST и $_COOKIE для полной уверенности в том, откуда берутся переменные.

Чтобы выполнить это, убедитесь, что соответствующая строка в файле *php.ini* выглядит следующим образом:

```
register_globals = Off
```

Что касается версии PHP 4.2, то в ней это значение устанавливается по умолчанию.

Обсуждение

Если параметр register_globals установлен в on, то внешние переменные, включая переменные из форм и cookies, импортируются прямо в глобальное пространство имен. Это очень удобно, но может привести к образованию брешей в безопасности, если вы не очень старательно проверяете переменные и место, где они определяются. Почему? Потому что может существовать переменная, с которой вы работаете внутри и не предполагаете ее использования снаружи, но ее значение было переписано без вашего ведома.

Ниже приведен простой пример. Есть страница, на которой вводится имя пользователя и пароль. Если они достоверны, то пользователю посылается его идентификационный номер и этот числовой идентификатор применяется для поиска и печати его персональной информации:

```
// предполагаем, что опция magic_quotes_gpc установлена в Off
$username = $dbh->quote($_GET['username']);
$password = $dbh->quote($_GET['password']);

$sth = $dbh->query("SELECT id FROM users WHERE username = $username AND
                    password = $password");

if (1 == $sth->numRows()) {
    $row = $sth->fetchRow(DB_FETCHMODE_OBJECT);
    $id = $row->id;
} else {
    "Print bad username and password";
}

if (!empty($id)) {
    $sth = $dbh->query("SELECT * FROM profile WHERE id = $id");
}
```

Обычно $id устанавливается вашей программой и представляет собою проверенный результат поиска в базе данных. Тем не менее, если кто-то изменяет строку GET и передает значение $id, в то время как параметр register_globals установлен в on, то даже после отрицательного результата поиска имени пользователя и пароля ваш сценарий выполнит второй запрос в базу данных и возвратит результат. Если regis-

ter_globals не установлен в on, то в $id не будет никакого значения, так как установлены только элементы $_REQUEST['id'] и $_GET['id'].

Конечно, существуют и другие способы решения этой проблемы даже при использовании параметра register_globals. Можно реструктурировать программу, чтобы исключить такую лазейку.

```
$sth = $dbh->query("SELECT id FROM users WHERE username = $username AND
                    password = $password");

if (1 == $sth->numRows()) {
    $row = $sth->fetchRow(DB_FETCHMODE_OBJECT);
    $id = $row->id;
    if (!empty($id)) {
        $sth = $dbh->query("SELECT * FROM profile WHERE id = $id");
    }
} else {
    "Print bad username and password";
}
```

Теперь вы используете переменную $id, только если она явным образом установлена в результате запроса базы данных. Однако иногда это трудно сделать из-за структуры программы. Другое решение состоит в том, чтобы вручную сбросить переменную с помощью функции unset() или инициализировать все переменные в начале сценария:

```
unset($id);
```

Таким образом, неправильное значение переменной $id удаляется до того, как оно сможет повлиять на вашу программу. Однако поскольку PHP не требует инициализировать переменную, можно забыть сделать это в одном месте; ошибка может вкрасться без предупреждения со стороны PHP.

См. также

Документацию по register_globals на *http://www.php.net/security.registerglobals.php*.

9.8. Пользовательские данные и escape-последовательности

Задача

Необходимо скрытно отображать на HTML-странице информацию, вводимую пользователем.

Решение

Для HTML-документа с внедренными ссылками и другими тегами, который нужно отображать как простой текст, используйте функцию htmlentities():

```
echo htmlentities('<p>O'Reilly & Associates</p>');
&lt;p&gt;O'Reilly & Associates&lt;/p&gt;
```

Обсуждение

В PHP есть пара функций для превращения в escape-последовательности символов в HTML-документе. Самая главная из них — функция `htmlspecialchars()`, преобразующая в escape-последовательности четыре символа: `<` `>` `"` и `&`. В зависимости от необязательных параметров она может также транслировать символ `'` вместо или в дополнение к `"`. Если надо закодировать что-то более сложное, используйте функцию `htmlentities()` — она расширяет возможности `htmlspecialchars()` до кодирования любого символа, который имеет элемент HTML.

```
$html = "<a href='fletch.html'>Stew's favorite movie.</a>\n";
print htmlspecialchars($html);              // двойные кавычки
print htmlspecialchars($html, ENT_QUOTES); // одинарные и двойные кавычки
print htmlspecialchars($html, ENT_NOQUOTES);    // ни те, ни другие
&lt;a href=&quot;fletch.html&quot;&gt;Stew's favorite movie.&lt;/a&gt;
&lt;a href=&quot;fletch.html&quot;&gt;Stew&#039;s favorite movie.&lt;/a&gt;
&lt;a href="fletch.html"&gt;Stew's favorite movie.&lt;/a&gt;
```

Обе функции допускают передачу им таблицы кодировки символов, которая устанавливает соответствие символов и элементов. Чтобы определить, какая таблица была использована предыдущей функцией, вызовите функцию `get_html_translation_table()` и передайте ей `HTML_ENTITIES` или `HTML_SPECIALCHARS`. Она возвращает массив, определяющий соответствие между символами и элементами; можно использовать его как основу для собственной таблицы.

```
$copyright = "Copyright © 2003 O'Reilly & Associates\n";
$table = get_html_translation_table(); // get <, >, ", and &
$table[©] = '&copy;'                   // add ©
print strtr($copyright, $table);
Copyright &copy; 2003 O'Reilly &amp; Associates
```

См. также

Рецепты 13.8, 18.20 и 10.7; документацию по функции `htmlentities()` на *http://www.php.net/htmlentities* и по функции `htmlspecialchars()` на *http://www.php.net/htmlspecialchars*.

9.9. Обработка внешних переменных с точками в именах

Задача

Необходимо обработать переменную с точкой в имени, но после того как форма представлена, не удается найти эту переменную.

Решение

Замените точку в имени переменной на символ подчеркивания. Например, если в форме находится элемент ввода с именем `foo.bar`, то доступ к нему в PHP осуществляется как к переменной `$_REQUEST['foo_bar']`.

Обсуждение

В PHP точка выступает в качестве оператора конкатенации строк, поэтому переменная формы, названная `animal.height`, автоматически преобразуется в переменную с именем `animal_height`, что позволяет избежать создания неопределенности для анализатора. Элементу `$_REQUEST['animal.height']` такая неопределенность не свойственна, но из соображений преемственности и совместимости преобразование происходит независимо от значения параметра `register_globals`.

Как правило, автоматическое преобразование имени переменной встречается при обработке изображений, используемых для представления формы. Допустим, у вас есть карта, оказывающая расположение вашего магазина, и вы хотите, чтобы пользователь щелкнул по ней для получения дополнительной информации. Приведем пример:

```
<input type="image" name="locations" src="locations.gif">
```

Когда пользователь щелкает по изображению, то координаты x и y передаются как переменные `locations.x` и `locations.y`. Поэтому в PHP для определения координат точки, на которой пользователь выполнил щелчок, нужно проверить элементы `$_REQUEST['locations_x']` и `$_REQUEST['locations_y']`.

С помощью ряда манипуляций можно создать переменную внутри PHP с точкой в имени:

```
${"a.b"} = 123; // принудительное приведение с помощью {}

$var = "c.d";   // косвенное именование переменной
$$var = 456;

print ${"a.b"} . "\n";
print $$var . "\n";
123
456
```

Обычно это не приветствуется из-за неудобного синтаксиса.

См. также

Документацию по внешним для PHP переменным на *http://www.php.net/language.variables.external.php*.

9.10. Использование элементов формы с несколькими вариантами значений

Задача

Есть элемент с несколькими значениями, такой как `checkbox` или `select`, но PHP видит только одно значение.

Решение

Вставьте квадратные скобки (`[ ]`) после имени переменной:

```
<input type="checkbox" name="boroughs[]" value="bronx"> The Bronx
<input type="checkbox" name="boroughs[]" value="brooklyn"> Brooklyn
<input type="checkbox" name="boroughs[]" value="manhattan"> Manhattan
<input type="checkbox" name="boroughs[]" value="queens"> Queens
<input type="checkbox" name="boroughs[]" value="statenisland"> Staten Island
```

В тексте программы рассматривайте эту переменную как массив:

```
print 'I love ' . join(' and ', $boroughs) . '!';
```

Обсуждение

Размещение квадратных скобок после имени переменной приказывает PHP считать переменную массивом, а не скалярной переменной. Когда PHP видит, что этой переменной присваивается другое значение, он автоматически увеличивает размер массива и размещает новое значение в конце этого массива. Выбор первых трех пунктов в разделе «Решение» был бы эквивалентен следующему коду в начале сценария:

```
$boroughs[] = "bronx";
$boroughs[] = "brooklyn";
$boroughs[] = "manhattan";
```

Следующий фрагмент можно использовать для получения из базы данных информации, которая отражает несколько записей:

```
foreach ($_GET['boroughs'] as $b) {
  $boroughs[] = strtr($dbh->quote($b),array('_' => '\_', '%' => '\%'));
}
$locations = join(',', $boroughs);

$dbh->query("SELECT address FROM locations WHERE borough IN ($locations)");
```

Этот синтаксис работает также с многомерными массивами:

```
<input type="checkbox" name="population[NY][NYC]" value="8008278">New York...
```

Если элемент отмечен, то он устанавливает элемент массива `$population['NY']['NYC']` в `8008278`.

Размещение квадратных скобок `[]` после имени переменной может вызвать проблемы в JavaScript при попытке обратиться к элементам. Вместо обращения к элементам по имени используйте числовой иден-

тификатор. Можно также поместить имя элемента в одинарные кавычки. Другой путь состоит в том, чтобы присвоить элементу идентификатор; можно взять имя без квадратных скобок [], а затем использовать этот идентификатор. Дано:

```
<form>
<input type="checkbox" name="myName[]" value="myValue" id="myName">
</form>
```

следующие три конструкции относятся к одному и тому же элементу формы:

```
document.forms[0].elements[0];            // использование числового
                                             идентификатора
document.forms[0].elements['myName[]'];   // использование имени
                                              в кавычках
document.forms[0].elements['myName'];     // использование идентификатора,
                                              назначенного вами
```

См. также

Более подробную информацию о массивах во введении в главе 4.

9.11. Создание выпадающих меню на основе текущей даты

Задача

Необходимо создать ряд выпадающих меню, которые автоматически привязываются текущей дате.

Решение

Для определения текущего времени во временной зоне веб-сервера вызовите функцию `date()` и выполните цикл по дням с помощью функции `mktime()`.

Следующий фрагмент программы генерирует значения элемента `option` для текущего дня и последующих шести дней. В данном случае текущий день – 1 января 2002 года.

```
list($hour, $minute, $second, $month, $day, $year) =
                                  split(':', date('h:i:s:m:d:Y'));

// печатаем последовательность дней одной недели
for ($i = 0; $i < 7; ++$i) {
    $timestamp = mktime($hour, $minute, $second, $month, $day + $i, $year);
    $date = date("D, F j, Y", $timestamp);

    print "<option value=\"$timestamp\">$date</option>\n";
}
<option value="946746000">Tue, January 1, 2002</option>
<option value="946832400">Wed, January 2, 2002</option>
```

```
<option value="946918800">Thu, January 3, 2002</option>
<option value="947005200">Fri, January 4, 2002</option>
<option value="947091600">Sat, January 5, 2002</option>
<option value="947178000">Sun, January 6, 2002</option>
<option value="947264400">Mon, January 7, 2002</option>
```

Обсуждение

В решении мы устанавливаем атрибут `value` для каждой даты в ее UNIX-представлении временной метки, поскольку считаем это более простым способом для нашей программы. Конечно, вы можете использовать любой формат, который вы находите более удобным и подходящим.

Не поддавайтесь искушению исключить вызов функции `mktime()`; даты и время не настолько совместимы, как вы думаете. В зависимости от ваших действий, можно получить не тот результат, на который вы рассчитываете. Например:

```
$timestamp = mktime(0, 0, 0, 10, 24, 2002); // October 24, 2002
$one_day = 60 * 60 * 24; // number of seconds in a day

// печатаем последовательность дней одной недели
for ($i = 0; $i < 7; ++$i) {
    $date = date("D, F j, Y", $timestamp);

    print "<option value=\"$timestamp\">$date</option>";

    $timestamp += $one_day;
}
<option value="972619200">Fri, October 25, 2002</option>
<option value="972705600">Sat, October 26, 2002</option>
<option value="972792000">Sun, October 27, 2002</option>
<option value="972878400">Sun, October 27, 2002</option>
<option value="972964800">Mon, October 28, 2002</option>
<option value="973051200">Tue, October 29, 2002</option>
<option value="973137600">Wed, October 30, 2002</option>
```

Этот сценарий должен напечатать месяц, день и год для семидневного периода, начиная с 24 октября 2002 года. Однако он работает не так, как ожидалось.

Почему здесь два воскресенья 27 октября 2002 года? Ответ: переход на летнее время (DST). Утверждение, что количество секунд в дне постоянно, неверное; в действительности почти наверняка оно изменится. Хуже всего то, что если вы не находитесь по времени рядом со сменой дат, то обязательно пропустите эту ошибку при тестировании.

См. также

Главу 3, в частности рецепт 3.12, а также рецепты 3.1, 3.2, 3.4, 3.10 и 3.13; документацию по функции `date()` на *http://www.php.net/date* и `mktime()` на *http://www.php.net/mktime*.

10

Доступ к базам данных

10.0. Введение

Базы данных являются ключевым моментом многих веб-приложений. База данных может хранить практически любую информацию, которую может потребоваться найти или обновить, например, список пользователей, каталог изделий или последние новости. Одна из причин, по которым PHP занимает такое значительное положение среди языков веб-программирования, заключается в его широком применении для поддержки баз данных. PHP может взаимодействовать (по последним подсчетам) с 17 различными базами данных – как с реляционными, так и нет. Реляционные базы, с которыми он может работать, – это DB++, FrontBase, Informix, Interbase, Ingres II, Microsoft SQL Server, mSQL, MySQL, Oracle, Ovrimos SQL Server, PostgreSQL, SESAM и Sybase. К нереляционным базам, относящимся к сфере действия PHP, относятся dBase, filePro, HyperWave, DBM-семейство баз данных, состоящих из плоских файлов. PHP также включает поддержку ODBC, поэтому даже если ваша любимая база данных не входит в приведенный выше список, то с ней можно работать при помощи средств PHP.

Если требуемые ресурсы для хранения данных невелики и нет необходимости обслуживать много пользователей, то, вероятно, простые текстовые файлы смогут заменить базу данных. Это обсуждается в рецепте 10.1. Для текстовых файлов не требуется специального программного обеспечения баз данных, но они подходят только для основных приложений, работающих с небольшой интенсивностью. Текстовые файлы не очень хороши для работы со структурированными данными; если данные подвержены значительным изменениям, то хранение их в простых файлах неэффективно по сравнению с базой данных.

Базы данных DBM, состоящие из плоских файлов, обсуждаемые в рецепте 10.2, обеспечивают большую устойчивость и эффективность, чем простые файлы, но все же ограничивают структуру данных парами ключ/значение. Они масштабируются лучше, чем обычные файлы,

особенно в случае данных только для чтения (или преимущественно для чтения).

Однако полностью возможности PHP раскрываются, когда он работает в паре с SQL-базой данных. Эта комбинация обсуждается в большинстве рецептов этой главы. SQL-базы данных могут быть сложными, но они отличаются исключительной мощностью. Для того чтобы использовать PHP с конкретной SQL-базой данных, надо во время компиляции PHP явно включить поддержку этой конкретной базы данных. Если PHP построен с поддержкой динамической загрузки модулей, то поддержка базы данных может быть также построена в виде динамического модуля.

В данной главе во многих примерах с SQL участвует таблица с информацией о знаках Зодиака. Ниже представлена структура этой таблицы:

```
CREATE TABLE zodiac (
  id INT UNSIGNED NOT NULL,
  sign CHAR(11),
  symbol CHAR(13),
  planet CHAR(7),
  element CHAR(5),
  start_month TINYINT,
  start_day TINYINT,
  end_month TINYINT,
  end_day TINYINT,
  PRIMARY KEY(id)
);
```

А это содержимое таблицы:[1]

```
INSERT INTO zodiac VALUES (1,'Aries','Ram','Mars','fire',3,21,4,19);
INSERT INTO zodiac VALUES (2,'Taurus','Bull','Venus','earth',4,20,5,20);
INSERT INTO zodiac VALUES (3,'Gemini','Twins','Mercury','air',5,21,6,21);
INSERT INTO zodiac VALUES (4,'Cancer','Crab','Moon','water',6,22,7,22);
INSERT INTO zodiac VALUES (5,'Leo','Lion','Sun','fire',7,23,8,22);
INSERT INTO zodiac VALUES (6,'Virgo','Virgin','Mercury','earth',8,23,9,22);
INSERT INTO zodiac VALUES (7,'Libra','Scales','Venus','air',9,23,10,23);
INSERT INTO zodiac VALUES
(8,'Scorpio','Scorpion','Mars','water',20,24,11,21);
INSERT INTO zodiac VALUES
(9,'Sagittarius','Archer','Jupiter','fire',11,22,12,21);
INSERT INTO zodiac VALUES
(10,'Capricorn','Goat','Saturn','earth',12,22,1,19);
INSERT INTO zodiac VALUES (11,'Aquarius','Water
Carrier','Uranus','air',1,20,2,18);
INSERT INTO zodiac VALUES
(12,'Pisces','Fishes','Neptune','water',2,19,3,20);
```

[1] Структура таблицы: номер, знак (sign), символ (symbol), планета (planet) и ее расшифровка. Для первой строки: `INSERT INTO zodiac VALUES (1,'Овен','баран','Марс','огонь',3,21,4,19);`.

Функции, необходимые для общения с различными базами данных, отличаются друг от друга, но каждая работает по сходному шаблону. Соединение с базой данных возвращает дескриптор соединения. Дескриптор соединения используется для создания операторных дескрипторов, которые ассоциируются с конкретными запросами. Впоследствии дескриптор оператора запроса получает результат запроса.

Приведенный ниже пример извлекает строки из таблицы `zodiac` базы данных Oracle с помощью интерфейса OCI8:

```
if (! $dbh = OCILogon('david', 'foo!bar','ORAINST')) {
    die("Can't connect: ".OCIError());
}

if (! $sth = OCIParse($dbh,'SELECT * FROM zodiac')) {
    die("Can't parse query: ".OCIError());
}

if (! OCIExecute($sth)) {
    die("Can't execute query: ".OCIError());
}

$cols = OCINumCols($sth);
while (OCIFetch($sth)) {
    for ($i = 1; $i <= $cols; $i++) {
        print OCIResult($sth,$i);
        print " ";
    }
    print "\n";
}
```

Функция `OCILogin()` соединяется с данным экземпляром Oracle с помощью имени пользователя и пароля. Можно опустить третий аргумент (экземпляр), если переменная окружения `ORACLE_SID` установлена в соответствии с требуемым экземпляром Oracle. Функция `OCIParse()` возвращает дескриптор оператора, а функция `OCIExecute()` выполняет запрос. При каждом вызове функции `OCIFetch()` очередная строка результата направляется в результирующий буфер. Значение в определенной колонке текущей строки доставляется в результирующий буфер функцией `OCIResult()`.

Ниже приведен тот же самый пример для PostgreSQL:

```
if (! $dbh = pg_connect('dbname=test user=david password=foo!bar')) {
    die("Can't connect: ".pg_errormessage());
}

if (! $sth = pg_exec($dbh,'SELECT * FROM zodiac')) {
    die("Can't execute query: ".pg_errormessage());
}

for ($i = 0, $j = pg_numrows($sth); $i < $j; $i++) {
    $ar = pg_fetch_row($sth,$i);
    foreach ($ar as $col) {
```

```
        print "$col ";
    }
    print "\n";
}
```

В данном случае функция pg_connect(), которой в качестве параметров передаются имя базы данных, имя пользователя и пароль, соединяется с PostgreSQL. Запрос выполняется функцией pg_exec(). Здесь нет необходимости в разделении этапов анализа и выполнения, как в случае с Oracle. Поскольку функция pg_fetch_row() размещает отдельные строки из результирующего множества в массиве, то выполняется цикл по всем строкам (при этом функция pg_numrows() вызывается для получения общего количества строк) и каждый элемент этого массива выводится на печать.

Приведем тот же самый пример, но для MySQL:

```
if (! $dbh = mysql_connect('localhost','david','foo!bar')) {
    die("Can't connect: ".mysql_error());
}

mysql_select_db('test');

if (! $sth = mysql_query('SELECT * FROM zodiac')) {
    die("Can't execute query: ".mysql_error());
}

while ($ar = mysql_fetch_row($sth)) {
    foreach ($ar as $col) {
        print "$col ";
    }
    print "\n";
}
```

Сначала функция mysql_connect(), в качестве параметров которой выступают имя хоста, имя пользователя и пароль, возвращает дескриптор базы данных. Затем вызывается функция mysql_select_db(), чтобы показать, какая база данных используется. Запрос выполняется функцией mysql_query(). Функция mysql_fetch_row() извлекает очередную строку, помещая ее в результирующее множество, и NULL, если строк больше нет; извлечение всех строк выполняется с помощью цикла while.

Каждый пример печатает все данные зодиакальной таблицы: по одной строке таблицы на каждой строке вывода, с пробелами между полями, как показано ниже:

```
Aries Ram Mars fire 3 21 4 19
Taurus Bull Venus earth 4 20 5 20
Gemini Twins Mercury air 5 21 6 21
Cancer Crab Moon water 6 22 7 22
Leo Lion Sun fire 7 23 8 22
Virgo Virgin Mercury earth 8 23 9 22
Libra Scales Venus air 9 23 10 23
```

```
Scorpio Scorpion Mars water 20 24 11 21
Sagittarius Archer Jupiter fire 11 22 12 21
Capricorn Goat Saturn earth 12 22 1 19
Aquarius Water Carrier Uranus air 1 20 2 18
Pisces Fishes Neptune water 2 19 3 20
```

Рецепты с 10.4 по 10.8 посвящены посылке запросов в базу данных и получению результатов в ответ, а также использованию запросов, изменяющих данные в базе.

PHP поддерживает множество параметров настройки и оптимизации для каждой базы данных. Большинство интерфейсов баз данных поддерживают постоянные соединения с помощью специальных функций. В предыдущих трех примерах можно было использовать функции `OCIPLogon()`, `pg_pconnect()` и `mysql_pconnect()` для создания постоянного соединения вместо однократного соединения для каждого обращения к сценарию.

Если требуется набор функций для определенной базы данных, то обратитесь к онлайновому руководству по PHP, соответствующие разделы которого содержат массу полезных советов по правильному конфигурированию и использованию для каждой базы данных. Начиная с рецепта 10.3 все SQL-примеры основаны на уровне абстракции базы данных PEAR DB, минимизирующем объем кода, который должен быть изменен для того, чтобы примеры работали с различными базами данных. Ниже приведен фрагмент программы, который может показать все строки из таблицы `zodiac`, используя DB и MySQL:

```
require 'DB.php';
$dbh = DB::connect('mysql://david:foo!bar@localhost/test');
$sth = $dbh->query('SELECT * FROM zodiac');
while ($row = $sth->fetchRow()) {
    print join(' ',$row).“\n“;
}
```

Единственное, что надо изменить, чтобы заставить этот код работать с другой базой данных, – это аргумент, переданный функции `DB::connect()` и определяющий, с какой базой данных нужно соединяться. Однако уровень абстракции базы данных не делает SQL полностью переносимым. Как правило, каждый производитель базы данных реализует свои индивидуальные расширения SQL, предоставляющие полезные возможности для одной базы данных и совершенно не работающие с другими базами.

Можно написать программу на SQL, которая будет работать на различных базах данных, не требуя больших изменений, но настройки базы данных, определяющие скорость и эффективность работы, перенести не удастся. Переносимый код для взаимодействия с базами данных может оказаться полезным, но необходимо соотнести затраты на разработку с вероятностью его применения на многих базах данных. Если программа разрабатывается для широкого распространения, то воз-

можность работы с многими базами данных это плюс. Однако если программа используется в рамках внутреннего проекта, то, вероятно, нет необходимости так сильно беспокоиться о независимости от базы данных.

С какой бы базой данных вы ни работали, скорее всего вы будете брать информацию из полей HTML-формы и сохранять эту информацию в базе данных. Некоторые символы, такие как апостроф и обратная косая черта, имеют особое назначение в SQL, поэтому следует соблюдать осторожность, если форма содержит такие символы. В PHP есть возможность, называемая «волшебные кавычки» (magic quotes), облегчающая работу с такими символами. Если параметр конфигурации `magic_quotes_gpc` установлен в `on`, то переменные, доставляемые запросами GET, запросами POST и cookies и содержащие одинарные кавычки, двойные кавычки, символы обратной косой черты и NULL, преобразовываются в escape-последовательности с помощью обратной косой черты. Можно также включить параметр `magic_quotes_runtime`, чтобы автоматически превращать в escape-последовательности символы кавычек, обратной косой черты и нуля, полученные из внешних источников, таких как запросы к базе данных или текстовые файлы. Так, если опция `magic_quotes_runtime` установлена в `on`, и вы читаете файл в массив с помощью функции `file()`, то специальные символы в этом массиве уже будут преобразованы в escape-последовательности.

Например, если элемент `$_REQUEST['excuse']` содержит «Ferris wasn't sick», а параметр `magic_quotes_gpc` установлен в `on`, то следующий запрос выполнится успешно:

```
$dbh->query("INSERT INTO excuses (truth) VALUES ('" . $_REQUESTS['excuse'] .
')');
```

Без волшебных кавычек апостроф в слове «wasn't» будет сигнализировать базе данных о конце строки, и запрос породит синтаксическую ошибку. Чтобы параметры `magic_quotes_gpc` и `magic_quotes_runtime` выделяли одинарные кавычки с помощью другой одинарной кавычки вместо символа обратной косой черты, установите параметр `magic_quotes_sybase` в `on`. Преобразование специальных символов в запросах в escape-последовательности обсуждается в рецепте 10.9. Основные приемы отладки, которые можно использовать для обработки ошибок, явившихся результатом выполнения запросов в базы данных, рассматриваются в рецепте 10.10.

Остальные рецепты посвящены задачам более сложным, чем простые запросы. В рецепте 10.11 показано, как автоматически сгенерировать уникальное значение идентификатора, которое можно использовать в качестве идентификатора записи. Конструирование запросов из списка полей во время исполнения объясняется в рецепте 10.12. Это позволяет легче справляться с запросами `INSERT` и `UPDATE`, затрагивающими много столбцов. В рецепте 10.13 продемонстрировано, как показывать ссылки, позволяющие путешествовать по результирующему множеству,

отображая небольшое количество записей на каждой странице. В рецепте 10.14 объясняется, как ускорить доступ к базе данных с помощью кэширования запросов и их результатов.

10.1. Работа с базами данных, состоящих из текстовых файлов

Задача

Требуется найти простой способ хранения информации в промежутках между выполнением запросов.

Решение

Используйте текстовый файл с необязательной блокировкой для предотвращения конфликтов. Можно хранить данные в любом подходящем формате (CSV, с разделителем – вертикальной чертой и т. д.) Один из удобных способов хранения данных состоит в их размещении в одной переменной (большом ассоциативном массиве), с последующим сохранением путем применения к этой переменной функции `serialize()`:

```
$data_file = '/tmp/data';

// открываем файл для чтения и записи
$fh = fopen($data_file,'a+') or die($php_errormsg);
rewind($fh)                  or die($php_errormsg);

// устанавливаем монопольную блокировку файла
flock($fh,LOCK_EX)           or die($php_errormsg);

// читаем и выполняем обратное преобразование данных
// из последовательной формы
$serialized_data = fread($fh,filesize($data_file)) or die($php_errormsg);
$data = unserialize($serialized_data);

/*
 * выполняем необходимые действия с данными
 */

// повторно переводим данные в последовательную форму
$serialized_data = serialize($data);

// очищаем файл
rewind($fh)                  or die($php_errormsg);
ftruncate($fp,0)             or die($php_errormsg);

// записываем данные обратно в файл и снимаем блокировку
if (-1 == (fwrite($fh,$serialized_data))) { die($php_errormsg); }
fflush($fh)                  or die($php_errormsg);
flock($fh,LOCK_UN)           or die($php_errormsg);
fclose($fh)                  or die($php_errormsg);
```

Обсуждение

Хранение данных в текстовом файле не требует инсталляции дополнительного программного обеспечения баз данных, но это практически единственное преимущество этого способа. Его главные недостатки – неповоротливость и неэффективность. Сначала необходимо заблокировать файл, а затем извлекать из него все данные подряд, даже если требуется лишь малая их часть. Пока блокировка не будет снята, все остальные процессы, а, следовательно, и все остальные пользователи будут вынуждены ждать, слоняясь без дела. Одно из ценных свойств базы данных состоит в том, что она предоставляет структурированный доступ к информации, что дает возможность блокировать (и загружать в память) только те данные, которые действительно требуются. Решение на основе текстовых файлов этого не позволяет.

Хуже то, что блокировка, которую можно применять к текстовым файлам, далеко не так устойчива, как блокировка в базе данных. Функция `flock()` осуществляет блокировку файлов, называемую необязательной, поэтому единственное, что может помешать многочисленным процессам передавить друг друга и испортить ваши данные, это хорошие манеры и аккуратное программирование. Но нет защиты от злоумышленных или добропорядочных, но плохо написанных программ.

См. также

Рецепт 5.7, в котором обсуждается сериализация данных; рецепт 18.24, детально объясняющий блокировку файлов; документацию по функции `flock()` на *http://www.php.net/flock*, по функции `serialize()` на *http://www.php.net/ serialize* и по функции `unserialize()` на *http://www.php.net/unserialize*.

10.2. Работа с базами данных DBM

Задача

Необходима более устойчивая и масштабируемая технология хранения простых данных, чем текстовых файлов.

Решение

Для доступа к базе данных типа DBM следует использовать уровень абстракции DBA:

```
$dbh = dba_open('fish.db','c','gdbm') or die($php_errormsg);

// извлекаем и модифицируем значения
if (dba_exists('flounder',$dbh)) {
  $flounder_count = dba_fetch('flounder',$dbh);
  $flounder_count++;
```

```
  dba_replace('flounder',$flounder_count);
  print "Updated the flounder count.";
} else {
  dba_insert('flounder',1);
  print "Started the flounder count.";
}

// больше нет ни одной тилапии
dba_delete('tilapia',$dbh);

// какая у нас рыбка?
for ($key = dba_firstkey($dbh);  $key !== false; $key = dba_nextkey($dbh)) {
   $value = dba_fetch($key);
   print "$key: $value\n";
}

dba_close($dbh);
```

Обсуждение

PHP способен поддерживать несколько различных типов машин баз данных DBM: GDBM, NDBM, DB2, DB3, DBM и CDB. Уровень абстракции DBA позволяет использовать одни и те же функции на любой машине DBM. Все машины хранят пары ключ/значение. Можно выполнять циклы по всем ключам базы данных, извлекать значение, связанное с конкретным ключом, и определять, есть ли определенный ключ. И ключи, и значения представляют собой строки.

Следующая программа поддерживает список имен пользователей и паролей с помощью базы данных DBM. Имя пользователя – это первый аргумент командной строки, а пароль – второй аргумент. Если имя пользователя уже существует в базе данных, то пароль заменяется данным паролем; в противном случае комбинация из имени пользователя и пароля добавляется в базу данных:

```
$user = $_SERVER['argv'][1];
$password = $_SERVER['argv'][2];

$data_file = '/tmp/users.db';

$dbh = dba_open($data_file,'c','gdbm') or die("Can't open db $data_file");

if (dba_exists($user,$dbh)) {
    print "User $user exists. Changing password.";
} else {
    print "Adding user $user.";
}

dba_replace($user,$password,$dbh) or die("Can't write
                                          to database $data_file");

dba_close($dbh);
```

Функция `dba_open()` возвращает дескриптор файла DBM (или `false` в случае ошибки). Она принимает три аргумента. Первый аргумент –

это имя файла DBM, а второй аргумент – режим открытия файла. Режим 'r' открывает доступ к существующей базе данных только на чтение, а 'w' открывает существующую базу данных на чтение и запись. Режим 'c' открывает доступ к базе данных на чтение/запись и создает базу данных, если она не существует. Последний режим, 'n', делает то же самое, что и режим 'c', но если база данных уже существует, то очищает ее. Третий аргумент функции dba_open() указывает используемый DBM-обработчик; в данном примере это 'gdbm'. Чтобы определить, какой DBM-обработчик был скомпилирован во время инсталляции PHP, загляните в секцию «DBA» вывода функции phpinfo(). Строка «Supported handlers» показывает то, что было выбрано.

Функция dba_exists() позволяет определить, есть ли такой ключ в базе данных DBM. Она принимает два аргумента: строку-ключ и дескриптор файла DBM. Она ищет ключ в файле DBM и возвращает true, если находит его (или false, если поиск неудачен). Функция dba_replace() принимает три аргумента: ключ-строку, строковое значение и дескриптор файла DBM. Она помещает ключ/значение в файл DBM. Если элемент с данным ключом уже существует, она заменяет этот элемент новым значением.

Для закрытия базы данных вызовите функцию dba_close(). Файл DBM, открытый функцией dba_open(), автоматически закрывается при завершении работы сценария, но необходимо явно вызвать функцию dba_close(), чтобы закрыть постоянные соединения, созданные функцией dba_popen().

С помощью функций dba_firstkey() и dba_nextkey() можно пройти в цикле по всем ключам в файле DBM, а функция dba_fetch() позволяет извлекать значения, связанные с каждым ключом. Приведенная ниже программа вычисляет общую длину всех паролей в файле DBM:

```
$data_file = '/tmp/users.db';
$total_length = 0;
if (! ($dbh = dba_open($data_file,'r','gdbm'))) {
    die("Can't open database $data_file");
}

$k = dba_firstkey($dbh);
while ($k) {
    $total_length += strlen(dba_fetch($k,$dbh));
    $k = dba_nextkey($dbh);
}

print "Total length of all passwords is $total_length characters.";

dba_close($dbh);
```

Функция dba_firstkey() инициализирует переменную $k значением первого ключа в файле DBM. При каждом прохождении цикла while функция dba_fetch() извлекает значение, соответствующее ключу $k, и переменная $total_length увеличивается на длину значения (вычисленного

посредством функции strlen()). С помощью функции dba_nextkey() переменной $k присваивается значение следующего ключа из файла.

Функция serialize() позволяет реализовать хранение сложных данных в файле DBM – точно так же, как это делается в случае текстового файла. Однако данные в файле DBM могут быть индексированы ключом:

```
$dbh = dba_open('users.db','c','gdbm') or die($php_errormsg);

// читаем данные и выполняем обратное преобразование
// из последовательной формы
if ($exists = dba_exists($_REQUEST['username'])) {
    $serialized_data = dba_fetch($_REQUEST['username'])
                           or die($php_errormsg);
    $data = unserialize($serialized_data);
} else {
    $data = array();
}

// обновляем значения
if ($_REQUEST['new_password']) {
    $data['password'] = $_REQUEST['new_password'];
}
$data['last_access'] = time();

// записываем данные обратно в файл
if ($exists) {
    dba_replace($_REQUEST['username'],serialize($data));
} else {
    dba_insert($_REQUEST['username'],serialize($data));
}

dba_close($dbh);
```

Несмотря на то что код этого примера может сохранять данные нескольких пользователей в одном файле, нельзя найти, например, время последней регистрации пользователя без выполнения цикла по всем ключам файла. Структурные данные такого типа относятся к базам данных SQL.

В некоторых областях каждый DBM-обработчик ведет себя по-разному. Например, GDBM предоставляет внутреннюю блокировку. Если один процесс открыл файл GDBM в режиме чтения/записи, то еще один вызов функции dba_open() для открытия того же самого файла в режиме чтения/записи закончится неудачей. Однако обработчик DB3 не предоставляет внутренней блокировки; для этого необходимо написать дополнительный код, как объясняется в рецепте 18.24 для текстовых файлов. Две DBA-функции также имеют особенности, связанные с типами баз данных: dba_optimize() и dba_sync(). Функция dba_optimize() вызывает специфическую для обработчика функцию оптимизации файла DBM. В настоящее время она реализована только для GDBM, при этом вызывается его функция gdbm_reorganize(). Функция

`dba_sync()` вызывает специфическую для обработчика функцию синхронизации файла DBM. В случае DB2 и DB3 вызывается их функция `sync()`. В случае GDBM вызывается его функция `gdbm_sync()`. Если применяются другие DBM-обработчики, то ничего не выполняется.

Использование базы данных DBM представляет собой шаг вперед по сравнению с текстовыми файлами, но при этом большинство возможностей SQL-баз данных недоступны. Структура данных ограничена парами ключ/значение, а устойчивость блокировки сильно зависит от DBM-обработчика. Все же выбор DBM-обработчиков может быть вполне оправдан, если требуется в основном только чтение данных при большой нагрузке; например, крупнейшая база данных кинофильмов в Интернете – сайт *http://www.imdb.com/* – основана на DBM.

См. также

Рецепт 5.7, в котором обсуждается сериализация данных; рецепт 18.24, в котором детально изучается блокировка; документацию по функциям DBA на *http://www.php.net/dba*; подробную информацию по DBM-обработчикам для баз данных DB2 и DB3 на *http://www.sleepycat.com/faq.html#program*; информацию по GDBM на *http://www.gnu.org/directory/gdbm.html* или на *http://www.mit.edu:8001/afs/athena.mit.edu/project/gnu/doc/html/gdbm_toc.html*; информацию по CDB на *http://cr.yp.to/cdb.html*; техническую спецификацию Internet Movie Database на *http://us.imdb.com/Help/Classes/Master/tech-info*.

10.3. Соединение с базой данных SQL

Задача

Необходимо получить доступ к SQL-базе данных.

Решение

Это делается при помощи метода `connect()` из PEAR DB:

```
require 'DB.php';

$dsn = 'mysql://david:foo!bar@localhost/test';

$dbh = DB::connect($dsn);
if (DB::isError($dbh)) { die ($dbh->getMessage()); }
```

Обсуждение

PEAR DB можно загрузить с сайта PEAR по адресу:

http://pear.php.net/package-info.php?package=DB

После загрузки DB-функций с *DB.php* соединитесь с базой данных посредством функции `DB::connect()`, выполните запрос с помощью метода

`$dbh->query()` и извлеките каждую строку с помощью метода `$sth->fetchRow()`. Пример в разделе «Решение» соединяется с базой данных MySQL. Чтобы соединиться с Oracle вместо MySQL, достаточно изменить значение переменной `$dsn`. Эта переменная содержит имя источника данных (DSN), строку, которая определяет, с какой базой и каким образом следует соединяться. Ниже приведено ее значение для Oracle:

```
$dsn = 'oci8://david:foo!bar@ORAINST';
```

Для базы данных PostgreSQL значение переменной `$dsn` равно:

```
$dsn = 'pgsql://david:foo!bar@unix(/tmp/.s.PGSQL.5432)/test';
```

DSN для PostgreSQL немного сложнее, поскольку оно определяет, что соединение должно быть выполнено через локальный сокет UNIX (имя пути к которому равно */tmp/.s.PGSQL.5432*), а не TCP/IP-соединение. Обычно имя источника данных имеет вид:

```
database_interface://user:password@hostname/database
```

Часть *database_interface* DSN представляет тип используемой базы данных, наример, Oracle, MySQL и т. д. В настоящее время PEAR поддерживает 10 машин баз данных, которые перечислены в табл. 10.1.

Таблица 10.1. Машины баз данных PEAR DB

Name	Database
fbsql	FrontBase
ibase	Interbase
ifx	Informix
msql	Mini-SQL
mssql	Microsoft SQL Server
mysql	MySQL
oci8	Oracle (использует интерфейс OCI8)
odbc	ODBC
pgsql	PostgreSQL
sybase	Sybase

Для использования конкретной машины баз данных PEAR DB необходимо собрать PHP с поддержкой базы данных, соответствующей выбранной машине. Обратите внимание, что для использования машины баз данных Oracle OCI8, в PHP необходимо включить расширение OCI8 (`--with-oci8` при компиляции). Старое Oracle-расширение PHP (`--with-oracle`) не совместимо с PEAR DB.

Строки *user* и *password* представляют имя пользователя и пароль, необходимые для соединения с базой данных. Строка *hostname* обычно пред-

ставляет собой имя хоста, на котором запущена база данных, но она может быть также именем экземпляра (для Oracle) или применявшимся ранее обозначением локального сокета с соблюдением специального синтаксиса. Строка *database* предназначена для хранения имени логической базы данных, такого, которое бы определялось в параметре *dbname* функции `pg_connect()` или в аргументе функции `mysql_select_db()`.

PEAR DB ни в коем случае не единственный уровень абстракции базы данных для PHP. Мы выбрали его только потому, что с ним легко работать и он широко распространен. Другие уровни абстракции базы данных включают ADOdb (*http://php.weblogs.com/ADODB*), Metabase (*http://en.static.phpclasses.org/browse.html/package/20.html*), класс DB_Sql в PHPLib (*http://phplib.sourceforge.net/*) и MDB (*http://pear.php.net/package-info.php?package=MDB*).

См. также

О выполнении запросов в SQL-базы данных в рецепте 10.4; рецепт 10.6, в котором рассказывается о модификации SQL-базы данных; о Pear DB на *http://pear.php.net/package-info.php?package=DB*; документацию по функции `DB::connect()` на *http://pear.php.net/manual/en/core.db.tut_connect.php* и *http://pear.php.net/manual/en/core.db.connect.php*; информацию по DSNs на *http://pear.php.net/manual/en/core.db.tut_dsn.php*.

10.4. Выполнение запросов к базе данных SQL

Задача

Необходимо извлечь некоторую информацию из базы данных.

Решение

Сначала вызовите функцию `DB::query()` из PEAR DB для посылки SQL-запроса в базу данных, а затем – функцию `DB_Result::fetchRow()` или функцию `DB_Result::fetchInto()` для извлечения каждой строки результата:

```
// использование функции fetchRow()
$sth = $dbh->query("SELECT sign FROM zodiac WHERE element LIKE 'fire'");
if (DB::isError($sth)) { die($sth->getMessage()); }

while($row = $sth->fetchRow()) {
    print $row[0]."\n";
}

// использование функции fetchInto()
$sth = $dbh->query("SELECT sign FROM zodiac WHERE element LIKE 'fire'");
if (DB::isError($sth)) { die($sth->getMessage()); }

while($sth->fetchInto($row)) {
    print $row[0]."\n";
}
```

Обсуждение

Метод fetchRow() возвращает данные, тогда как метод fetchInto() помещает данные в переменную, которую ему передают. И метод fetchRow(), и метод fetchInto() возвращают NULL, если больше нет ни одной строки. Если любой из двух методов сталкивается с ошибкой при извлечении строки, то он возвращает объект DB_Error точно так же, как это делают методы DB::connect() и DB::query(). Можно вставить проверку этой ситуации внутри цикла:

```
while($row = $sth->fetchRow()) {
    if (DB::isError($row)) { die($row->getMessage()); }
    print $row[0]."\n";
}
```

Если параметр magic_quotes_gpc установлен в on, то можно использовать переменную формы непосредственно в запросе:

```
$sth = $dbh->query(
    "SELECT sign FROM zodiac WHERE element LIKE '" . $_REQUEST['element'] . "'");
```

Если нет, то надо преобразовать значение с помощью функции DB::quote() или использовать символ-заместитель:

```
$sth = $dbh->query("SELECT sign FROM zodiac WHERE element LIKE " .
                   $dbh->quote($_REQUEST['element']));

$sth = $dbh->query('SELECT sign FROM zodiac WHERE element LIKE ?',
                   array($_REQUEST['element']));
```

В рецепте 10.9 подробно рассказано, когда следует брать значения в кавычки и как это делать.

По умолчанию методы fetchRow() и fetchInto() размещают информацию в числовых массивах. Но можно сохранять данные в ассоциативных массивах или объектах, передавая методам дополнительные параметры. В случае ассоциативных массивов задается параметр DB_FETCHMODE_ASSOC:

```
while($row = $sth->fetchRow(DB_FETCHMODE_ASSOC)) {
    print $row['sign']."\n";
}

while($sth->fetchInto($row,DB_FETCHMODE_ASSOC)) {
    print $row['sign']."\n";
}
```

Для объектов указывается параметр DB_FETCHMODE_OBJECT:

```
while($row = $sth->fetchRow(DB_FETCHMODE_OBJECT)) {
    print $row->sign."\n";
}

while($sth->fetchInto($row,DB_FETCHMODE_OBJECT)) {
    print $row->sign."\n";
}
```

Независимо от режима выборки методы по-прежнему возвращают NULL, если не осталось возвращаемых данных, и объект DB_Error в случае ошибки. Режим числового массива по умолчанию может быть установлен с помощью параметра DB_FETCHMODE_ORDERED. Можно установить режим выборки, который будет использоваться во всех последующих вызовах методов fetchRow() или fetchInto(), с помощью метода DB::setFetchMode():

```
$dbh->setFetchMode(DB_FETCHMODE_OBJECT);

while($row = $sth->fetchRow()) {
    print $row->sign."\n";
}

// последующие запросы и вызовы метода fetchRow() также возвращают объекты
```

См. также

Рецепт 10.3 о выполнении соединений с SQL-базой данных; рецепт 10.6 о модификации SQL-базы данных; рецепт 10.9, в котором детально объясняется, как заключать данные в кавычки для их безопасного включения в запросы; документацию по методу DB::query() на *http://pear.php.net/manual/en/core.db.tut_query.php* и *http://pear.php.net/manual/en/core.db.query.php*, по выполнению выборки на *http://pear.php.net/manual/en/core.db.tut_fetch.php*, по методу DB_Result::fetchRow() на *http://pear.php.net/manual/en/core.db.fetchrow.php*, по методу DB_Result::fetchInto() на *http://pear.php.net/manual/en/core.db.fetchinto.php* и по методу DB::setFetchMode() на *http://pear.php.net/manual/en/core.db.setfetchmode.php*.

10.5. Извлечение строк без цикла

Задача

Необходимо найти короткий путь выполнения запросов и извлечения данных.

Решение

В случае PEAR DB для извлечения первой (или единственной) строки из запроса используйте метод DB::getRow():

```
$row = $dbh->getRow("SELECT planet,symbol FROM zodiac
                     WHERE sign LIKE 'Pisces'");
```

Для извлечения всех строк из запроса применяется метод DB::getAll():

```
$rows = $dbh->getAll("SELECT planet,symbol FROM zodiac WHERE element LIKE
'fire'");
```

Для того чтобы получить только один столбец одной строки, применяется метод DB::getOne():

```
$col = $dbh->getOne("SELECT symbol FROM zodiac WHERE sign = 'Libra'");
```

Метод DB::getCol() позволяет получить один столбец из всех строк:

```
$cols = $dbh->getCol('SELECT symbol FROM zodiac');
```

Метод DB::getAssoc() предназначен для извлечения всех строк запроса и размещения их в ассоциативном массиве, индексированном по первому столбцу запроса:

```
$assoc = $dbh->getAssoc(
    "SELECT sign,symbol,planet FROM zodiac WHERE element LIKE 'water'");
```

Обсуждение

Все эти функции возвращают объект DB_Error, если возникает ошибка во время выполнения запроса или извлечения данных. Если запрос не возвращает результатов, то методы getRow() и getOne() возвращают NULL; методы getAll(), getCol() и getAssoc() возвращают пустой массив.

Во время получения результатов метод getRow() возвращает массив или объект в зависимости от текущего режима выборки. Метод getAll() — массив массивов или массив объектов, также в зависимости от режима выборки. Единственным результатом, который возвращает метод getOne(), обычно бывает строка, поскольку драйверы базы данных PHP, как правило, приводят извлеченные результаты к строковому типу. Аналогично метод getCol() возвращает массив результатов, при этом значения обычно представляют собой строки. Метод getAssoc() возвращает результаты в виде массива. Тип элементов этого массива определяется режимом выборки.

Как и в случае с методом DB::query(), можно передать в эти функции запрос, содержащий символы-заместители, и массив параметров, для заполнения этих символов. Эти параметры соответствующим образом заключаются в кавычки, когда они ставятся в запрос вместо символа-заместителя:

```
$row = $dbh->getRow('SELECT planet,symbol FROM zodiac WHERE sign LIKE ?',
                    array('Pisces'));
```

Параметр array является вторым аргументом каждой из этих функций, за исключением методов getCol() и getAssoc(). Для этих двух функций данный параметр является третьим аргументом. Второй аргумент метода getCol() содержит номер возвращаемого столбца, если не требуется первый столбец (номер столбца 0). Например, следующее выражение возвращает значение столбца planet:

```
$cols = $dbh->getCol('SELECT symbol,planet FROM zodiac',1);
```

Вторым аргументом функции getAssoc() является логическое значение, сообщающее функции, надо ли превращать значения возвращаемого ею ассоциативного массива в массивы, даже если это скалярные величины. Рассмотрим в качестве примера следующий запрос:

```
$assoc = $dbh->getAssoc(
    "SELECT sign,symbol FROM zodiac WHERE element LIKE 'water'");
print_r($assoc);
Array
(
    [Cancer] => Crab
    [Scorpio] => Scorpion
    [Pisces] => Fishes
)
```

Запрос, переданный методу getAssoc(), извлекает лишь два столбца: первый – это ключ массива, а второй – скалярное значение массива. Ниже показано, как принудительно перевести значения массива в одноэлементные массивы:

```
$assoc = $dbh->getAssoc(
    "SELECT sign,symbol FROM zodiac WHERE element LIKE 'water'",true);
print_r($assoc);
Array
(
    [Cancer] => Array
        (
            [0] => Crab
        )
    [Scorpio] => Array
        (
            [0] => Scorpion
        )
    [Pisces] => Array
        (
            [0] => Fishes
        )
)
```

Точно так же, как это делают методы fetchRow() и fetchInto(), методы getRow(), getAssoc() и getAll() по умолчанию размещают данные в числовых массивах. Можно передать им режим выборки (третий аргумент функций getRow() или getAll(), четвертый аргумент функции getAssoc()). Они также соблюдают режим выборки, установленный методом DB::setFetchMode().

См. также

Более подробное описание режима выборки в рецепте 10.4; документацию по выборке на *http://pear.php.net/manual/en/core.db.tut_fetch.php*, по функции DB::getRow() на *http://pear.php.net/manual/en/core.db.ge-*

trow.php, по функции `DB::getAll()` на *http://pear.php.net/manual/en/core.db.getall.php*, по функции `DB::getOne()` на *http://pear.php.net/manual/en/core.db.getone.php*, по функции `DB::getCol()` на *http://pear.php.net/manual/en/core.db.getcol.php* и по функции `DB::getAssoc()` на *http://pear.php.net/manual/en/core.db.getassoc.php*.

10.6. Модификация данных в базе данных SQL

Задача

Необходимо добавлять, удалять или изменять данные в SQL-базе данных.

Решение

Для посылки запросов `INSERT`, `DELETE` или `UPDATE` в PEAR DB предназначена функция `DB::query()`:

```
$dbh->query("INSERT INTO family (id,name) VALUES (1,'Vito')");

$dbh->query("DELETE FROM family WHERE name LIKE 'Fredo'");

$dbh->query("UPDATE family SET is_naive = 1 WHERE name LIKE 'Kay'");
```

Можно также подготовить запрос посредством функции `DB::prepare()` и выполнить его, вызвав функцию `DB::execute()`:

```
$prh = $dbh->prepare('INSERT INTO family (id,name) VALUES (?,?)');
$dbh->execute($prh,array(1,'Vito'));

$prh = $dbh->prepare('DELETE FROM family WHERE name LIKE ?');
$dbh->execute($prh,array('Fredo'));

$prh = $dbh->prepare('UPDATE family SET is_naive = ? WHERE name LIKE ?');
$dbh->execute($prh,array(1,'Kay');
```

Обсуждение

Метод `query()` посылает в базу данных все, что ему передают, поэтому он может применяться в запросах получения данных и в запросах модификации данных.

Методы `prepare()` и `execute()` особенно полезны в запросах, которые требуется выполнить несколько раз. Подготовленный запрос может быть исполнен без повторной подготовки:

```
$prh = $dbh->prepare('DELETE FROM family WHERE name LIKE ?');
$dbh->execute($prh,array('Fredo'));
$dbh->execute($prh,array('Sonny'));
$dbh->execute($prh,array('Luca Brasi'));
```

См. также

Рецепт 10.3 о соединении с SQL-базой данных; рецепт 10.4 о выполнении запросов в SQL-базу данных; рецепт 10.7, в котором подробно обсуждаются методы `prepare()` и `execute()`; документацию по методу `DB::query()` на *http://pear.php.net/manual/en/core.db.query.php*, по методу `DB::prepare()` на *http://pear.php.net/manual/en/core.db.prepare.php* и по методу `DB::execute()` на *http://pear.php.net/manual/en/core.db.execute.php*.

10.7. Эффективное повторение запросов

Задача

Необходимо несколько раз повторить выполнение одного и того же запроса, каждый раз подставляя новые значения.

Решение

В PEAR DB определите запрос с помощью функции `DB::prepare()`, а затем выполните запрос, вызвав функцию `DB::execute()`. Символы-заместители в запросе, переданные в функцию `prepare()`, замещаются данными функцией `execute()`:

```
$prh = $dbh->prepare("SELECT sign FROM zodiac WHERE element LIKE ?");

$sth = $dbh->execute($prh,array('fire'));
while($sth->fetchInto($row)) {
    print $row[0]."\n";
}

$sth = $dbh->execute($prh,array('water'));
while($sth->fetchInto($row)) {
    print $row[0]."\n";
}
```

Обсуждение

Первая функция `execute()` из раздела «Решение» начинает выполнение запроса:

```
SELECT sign FROM zodiac WHERE element LIKE 'fire'
```

Вторая запускает запрос:

```
SELECT sign FROM zodiac WHERE element LIKE 'water'
```

В каждом случае функция `execute()` заменяет символ-заместитель `?` на значение своего второго аргумента. Если символов-заместителей более одного, то аргументы надо разместить в массиве в порядке их появления в запросе:

```
$prh = $dbh->prepare(
    "SELECT sign FROM zodiac WHERE element LIKE ? OR planet LIKE ?");

// SELECT sign FROM zodiac WHERE element LIKE 'earth' OR planet LIKE 'Mars'
$sth = $dbh->execute($prh,array('earth','Mars'));
```

Значения, подставляемые вместо символов-заместителей, заключаются в кавычки. Чтобы вставить содержимое файла, используйте символ-заместитель & и передайте функции execute() имя файла:

```
/* Структура таблицы изображений:
   CREATE TABLE pictures (
       mime_type CHAR(20),
       data      LONGBLOB
   )
*/

$prh = $dbh->prepare('INSERT INTO pictures (mime_type,data) VALUES (?,&)');
$sth = $dbh->execute($prh,array('image/jpeg','test.jpeg'));
```

Для того чтобы функция execute() не заключала значения в кавычки, надо задать параметр !. Этот способ может быть небезопасен, если применяется для пользовательского ввода; но он удобен, если значение представляет собой не скалярную величину, а функцию базы данных. Так, в приведенном ниже запросе функция NOW() нужна для того, чтобы вставить текущие дату и время в столбец DATETIME:

```
$prh = $dbh->prepare("INSERT INTO warnings (message,message_time) VALUES
(?,!)");
$dbh->execute($prh,array("Don't cross the streams!",NOW()));
```

Для многократного выполнения подготовленного оператора с различными аргументами предназначена функция executeMultiple(). Вместо простой передачи одного массива аргументов, как при вызове функции execute(), в данном случае передается массив массивов аргументов:

```
$prh = $dbh->prepare('INSERT INTO pictures (mime_type,data) VALUES (?,&)');

$ar = array(array('image/jpeg','earth.jpeg'),
            array('image/gif','wind.gif'),
            array('image/jpeg','fire.jpeg'));

$sth = $dbh->executeMultiple($prh,$ar);
```

Необходимо сначала объявить массив, а затем передать его функции executeMultiple(), в противном случае PHP выдает сообщение об ошибке, в котором говорится, что параметр функции executeMultiple()передан по ссылке. Функция executeMultiple() выполняет цикл по всем аргументам в массиве, но если в процессе прохождения встречается ошибка, функция не будет продолжать обработку остальных аргументов. Если все запросы успешны, то функция executeMultiple() возвращает константу DB_OK. Функция executeMultiple() никогда не возвращает результирующий объект, поэтому ее нельзя применять в запросах, возвращающих данные.

Машины баз данных Interbase и OCI8 могут использовать возможности родных баз данных, поэтому для запросов INSERT/UPDATE/DELETE пара методов prepare()/execute() более эффективна, чем функция query(). Машина Interbase использует функции ibase_prepare() и ibase_execute(), а машина OCI8 использует функции OCIParse(), OCIBindByName() и OCIExecute(). Другие машины баз данных конструируют запросы с помощью интерполяции значений, предоставленных для использования вместо символов-заместителей.

См. также

Документацию по функции DB::prepare() на *http://pear.php.net/manual/en/core.db.prepare.php*, по функции DB::execute() на *http://pear.php.net/manual/en/core.db.execute.php* и по функции DB:: executeMultiple() на *http://pear.php.net/manual/en/core.db.executemultiple.php*; обзор по выполнению запросов на *http://pear.php.net/manual/en/core.db.tut_execute.php*.

10.8. Определение количества строк, возвращенных запросом

Задача

Требуется узнать, какое количество строк возвратил запрос SELECT, или сколько строк были изменены запросом INSERT, UPDATE или DELETE.

Решение

Количество строк, возвращенных запросом SELECT, определяется с помощью метода PEAR DB DB_Result:: numRows():

```
// запрос
$sth = $dbh->query('SELECT * FROM zodiac WHERE element LIKE ?',
array('water'));
$water_rows = $sth->numRows();

// подготавливаем и выполняем
$prh = $dbh->prepare('SELECT * FROM zodiac WHERE element LIKE ?');
$sth = $dbh->execute($prh,array('fire'));
$fire_rows = $sth->numRows();
```

Для определения количества строк, измененных запросом INSERT, UPDATE или DELETE, применяется метод DB::affectedRows():

```
$sth = $dbh->query('DELETE FROM zodiac WHERE element LIKE ?',array('fire'));
$deleted_rows = $dbh->affectedRows();

$prh = $dbh->prepare('INSERT INTO zodiac (sign,symbol) VALUES (?,?)',
                     array('Leap Day','Kangaroo'));
$dbh->execute($prh,$sth);
```

```
$inserted_rows = $dbh->affectedRows();

$dbh->query('UPDATE zodiac SET planet = ? WHERE sign LIKE ?',
            array('Trantor','Leap Day'));
$updated_rows = $dbh->affectedRows();
```

Обсуждение

Количество строк в результирующем множестве — свойство этого множества, поэтому метод `numRows()` применяется к спецификатору оператора, а не базы данных. Однако количество строк, задействованных в запросе обработки, не может быть свойством результирующего множества, поскольку такие запросы не возвращают результатов. Как следствие, функция `affectedRows()` является методом спецификатора базы данных.

См. также

Документацию по методу `DB_Result::numRows()` на *http://pear.php.net/manual/en/core.db.numrows.php* и по методу `DB::affectedRows()` на *http://pear.php.net/manual/en/core.db.affectedrows.php*.

10.9. Преобразование кавычек в escape-последовательности

Задача

Необходимо сделать текстовые или двоичные данные безопасными для запросов.

Решение

Напишите все запросы с символами-заместителями и передайте значения в массиве для замещения этих символов:

```
$sth = $dbh->query('UPDATE zodiac SET planet = ? WHERE id = 2',
                   array('Melmac'));

$rows = $dbh->getAll('SELECT * FROM zodiac WHERE planet LIKE ?',
                     array('M%'));
```

Для преобразования специальных символов в escape-последовательности и для того, чтобы быть уверенным в том, что строки соответствующим образом отмечены (обычно с помощью одинарных кавычек вокруг них), можно также использовать метод PEAR DB `DB::quote()`:

```
$planet = $dbh->quote($planet);
$dbh->query("UPDATE zodiac SET planet = $planet WHERE id = 2");
```

Если значение переменной `$planet` равно `Melmac`, то метод `$dbh->quote($planet)` при использовании MySQL возвращает строку `'Melmac'`. Ес-

ли значение переменной `$planet` равно `Ork's Moon`, то метод `$dbh->quote($planet)` возвращает `'Ork\'s Moon'`.

Обсуждение

Метод `DB::quote()` гарантирует, что текстовые или двоичные данные соответствующим образом заключены в кавычки, но также необходимо заключить в кавычки групповые символы SQL `%` и `_`, чтобы обеспечить возвращение оператором `SELECT` правильного результата. Если переменная `$planet` установлена в `Melm%`, то этот запрос возвращает строки, у которых значение столбца `planet` равно `Melmac`, `Melmacko`, `Melmacedonia` или какому-нибудь другому, начинающемуся со строки `Melm`:

```
$planet = $dbh->quote($planet);
$dbh->query("SELECT * FROM zodiac WHERE planet LIKE $planet");
```

Поскольку `%` – это групповой символ SQL, означающий «любое количество символов» (подобно символу `*` при замене имен в оболочке), а символ подчеркивания `_` это групповой символ SQL, означающий «один символ» (подобно символу `?` при замене имен в оболочке), их необходимо также преобразовать в escape-последовательности с помощью символа обратной косой черты. Для их преобразования в escape-последовательности применяется функция `strtr()`:

```
$planet = $dbh->quote($planet);
$planet = strtr($planet,array('_' => '\_', '%' => '\%'));
$dbh->query("SELECT * FROM zodiac WHERE planet LIKE $planet");
```

Метод `strtr()` должен быть вызван после вызова метода `DB::quote()`. В противном случае метод `DB::quote()` преобразует в escape-последовательности и символы обратной косой черты, добавленные функцией `strtr()`. Когда метод `DB::quote()` вызывается первым, строка `Melm_` превращается в строку `Melm\_`, которая интерпретируется базой данных как «строка Melm, за которой следует буквенный символ подчеркивания». Если метод `DB::quote()` вызывается вслед за методом `strtr()`, то строка `Melm_` превращается в строку `Melm\\_`, интерпретируемую базой данных как «строка Melm, за которой следует буквенный символ обратной косой черты, за которым следует групповой символ подчеркивания».

Метод кавычек определен в базовом классе DB, но некоторые, специфичные для баз данных подклассы, переопределяют этот метод, чтобы обеспечить соответствующее применение кавычек для конкретной базы данных. Применение метода `DB::quote()` вместо замещения специальных символов делает программу более переносимой.

Заключение в кавычки символов-заместителей происходит, даже если параметры `magic_quotes_gpc` или `magic_quotes_runtime` установлены в `on`. Аналогично, если к значению применяется метод `DB:quote()`, когда волшебные кавычки активны, то значение все равно заключается в кавычки. Для максимальной переносимости программы перед выполне-

нием запроса или вызовом метода DB::quote() удалите снабженные магическими кавычками символы обратной косой черты:

```
$fruit = ini_get('magic_quotes_gpc') ? stripslashes($_REQUEST['fruit']) :
    $_REQUEST['fruit'];

$dbh->query('UPDATE orchard SET trees = trees - 1 WHERE fruit LIKE ?',
            array($fruit));
```

См. также

Документацию по методу DB::quote() на *http://pear.php.net/manual/en/core.db.quote.php* и по магическим кавычкам на *http://www.php.net/manual/en/ref.info.php#ini.magic-quotes-gpc*.

10.10. Регистрация отладочной информации и ошибок

Задача

Необходимо получить доступ к информации, помогающей в устранении проблем. Например, если запрос завершен неудачно, то требуется просмотреть сообщения об ошибках, возвращенных базой данных.

Решение

Для исследования результатов одиночного запроса применяется метод DB::isError():

```
$sth = $dbh->query("SELECT aroma FROM zodiac WHERE element LIKE 'fire'");
DB::isError($sth) and print 'Database Error: '.$sth->getMessage();
```

Метод DB::setErrorHandling() позволяет предусмотреть автоматическое реагирование на любую ошибку базы данных:

```
$dbh->setErrorHandling(PEAR_ERROR_PRINT);
$sth = $dbh->query("SELECT aroma FROM zodiac WHERE element LIKE 'fire'");
```

Обсуждение

Большинство методов PEAR DB, столкнувшись с ошибкой, возвращают объект DB_Error. Метод DB::isError() возвращает значение true, если ему передан объект DB_Error, поэтому его можно использовать для тестирования результатов отдельных запросов. Класс DB_Error является дочерним классом класса PEAR::Error, поэтому для отображения информации об ошибке можно применять такие методы, как getMessage(). Все содержимое объекта Error можно вывести при помощи функции print_r():

```
$sth = $dbh->query('SELECT aroma FROM zodiac WHERE element LIKE 'fire'");
if (DB::isError($sth)) {
    print_r($sth);
}
```

В таблице zodiac нет столбца aroma, поэтому в результате будет напечатано:

```
db_error Object
(
    [error_message_prefix] =>
    [mode] => 1
    [level] => 1024
    [code] => -19
    [message] => DB Error: no such field
    [userinfo] => SELECT aroma FROM zodiac WHERE element LIKE 'fire' \
[nativecode=1054 ** Unknown column 'aroma' in 'field list']
    [callback] =>
)
```

Применение функции setErrorHandling() позволяет определить действия, автоматически выполняемые всякий раз, когда возникает ошибка базы данных. Укажите функции setErrorHandling() образ действий, передав ей константу PEAR_ERROR. Константа PEAR_ERROR_PRINT инициирует печать сообщения об ошибке, но выполнение программы продолжается:

```
$dbh->setErrorHandling(PEAR_ERROR_PRINT);
$sth = $dbh->query("SELECT aroma FROM zodiac WHERE element LIKE 'fire'");
```

В результате будет напечатано:

```
DB Error: no such field
```

Для того чтобы напечатать сообщение об ошибке и выйти из программы, используйте константу PEAR_ERROR_DIE. Или константу PEAR_ERROR_CALLBACK для запуска пользовательской функции при возникновении ошибки. Такая пользовательская функция может напечатать даже более подробную информацию:

```
function pc_log_error($error_obj) {
    error_log(sprintf("%s (%s)",$error_obj->message,$error_obj->userinfo));
}

$dbh->setErrorHandling(PEAR_ERROR_CALLBACK,'pc_log_error');
$sth = $dbh->query("SELECT aroma FROM zodiac WHERE element LIKE 'fire'");
```

Когда некорректный SQL-оператор в методе $dbh->query() становится причиной ошибки, то вызывается функция pc_log_error() с переданным ей в качестве аргумента объектом DB_Error. Функция обратного вызова pc_log_error() использует свойства объекта DB_Error для вывода более полного сообщения в журнал ошибок:

```
DB Error: no such field (SELECT aroma FROM zodiac WHERE element
LIKE 'fire' [nativecode=Unknown column 'aroma' in 'field list'])
```

Для сбора всей информации из объекта ошибки и записи ее в журнал ошибок применяют функцию print_r() и буферизацию вывода при обработке ошибки:

```
function pc_log_error($error_obj) {
    ob_start();
    print_r($error_obj);
    $dump = ob_get_contents();
    ob_end_clean();
    error_log('Database Error: '.$dump);
}

$dbh->setErrorHandling(PEAR_ERROR_CALLBACK,'pc_log_error');
$sth = $dbh->query("SELECT aroma FROM zodiac WHERE element LIKE 'fire'");
```

Следующий фрагмент включает все поля объекта ошибки в журнал сообщений об ошибках:

```
Database Error: db_error Object
(
    [error_message_prefix] =>
    [mode] => 16
    [level] => 1024
    [code] => -19
    [message] => DB Error: no such field
    [userinfo] => SELECT aroma FROM zodiac WHERE element LIKE 'fire' \
[nativecode=1054 ** Unknown column 'aroma' in 'field list']
    [callback] => pc_log_error
)
```

С помощью константы PEAR_ERROR_TRIGGER можно также заставить объект DB_Error генерировать внутреннюю ошибку PHP:

```
$dbh->setErrorHandling(PEAR_ERROR_TRIGGER);
$sth = $dbh->query("SELECT aroma FROM zodiac WHERE element LIKE 'fire'");
```

С константой PEAR_ERROR_TRIGGER функция setErrorHandling() для генерации внутренней ошибки использует функцию PHP trigger_error(). К этой ошибке применяется обработчик ошибок PHP по умолчанию или определенный пользователем обработчик, назначенный функцией set_error_handler(). По умолчанию внутренней ошибкой является E_USER_NOTICE:

```
<br />
<b>Notice</b>:  DB Error: no such field in <b>/usr/local/lib/php/PEAR.php</b>
\
on line <b>593</b><br />
```

Ошибки E_USER_WARNING или E_USER_ERROR воспроизводятся с помощью передачи второго аргумента функции setErrorHandling():

```
$dbh->setErrorHandling(PEAR_ERROR_TRIGGER,E_USER_ERROR);
$sth = $dbh->query("SELECT aroma FROM zodiac WHERE element LIKE 'fire'");
```

При возникновении ошибки E_USER_ERROR выполнение программы прекращается после выдачи следующего сообщения об ошибке:

```
<br />
```

```
<b>Fatal error</b>:  DB Error: no such field in <b>/usr/local/lib/php/
PEAR.php</b>
on line <b>593</b><br />
```

См. также

Рецепт 8.12, в котором обсуждается буферизация вывода; рецепты с 8.15 по 8.17, в которых рассказывается об обработке ошибок и создании пользовательского обработчика ошибок; документацию по функции `DB::isError()` на *http://pear.php.net/manual/en/core.db.iserror.php*, по классу `PEAR_Error` на *http://pear.php.net/manual/en/class.pear-error.php*, по функции `trigger_error()` на *http://www.php.net/trigger-error* и по функции `set_error_handler()` на *http://www.php.net/set-error-handler*.

10.11. Автоматическое присваивание уникальных значений идентификаторов

Задача

Необходимо создать возрастающую последовательность уникальных идентификаторов – целых чисел. Например, требуется присвоить уникальные идентификаторы пользователям, договорам или другим объектам при внесении их в базу данных.

Решение

В PEAR DB для получения следующего целого значения применяется функция `DB::nextId()` с именем последовательности:

```
$id = $dbh->nextId('user_ids');
```

Обсуждение

По умолчанию последовательность создается, если она еще не существует, и первому идентификатору в последовательности присваивается `1`. В следующем операторе `INSERT` можно использовать целое значение, возвращенное функцией `nextId()`:

```
$id = $dbh->nextId('user_ids');
$dbh->query("INSERT INTO users (id,name) VALUES ($id,'david')");
```

Этот оператор вставляет запись в таблицу `users` с `id`, равным `1`, и `name`, равным `david`. Чтобы предотвратить создание последовательности, если она не существует, передайте значение `false` в качестве второго аргумента в функцию `nextId()`:

```
$id = $dbh->nextId('user_ids',false);
$dbh->query("INSERT INTO users (id,name) VALUES ($id,'david')");
```

Для создания последовательности вызовите функцию `createSequence()`; а для удаления последовательности – функцию `dropSequence()`:

```
$dbh->createSequence('flowers');
$id = $dbh->nextId('flowers');
$dbh->dropSequence('flowers');
```

При попытке создания последовательности, которая уже существует, или удаления несуществующей последовательности возвращается объект DB_Error.

См. также

Документацию по функции DB::nextId() на *http://pear.php.net/manual/en/core.db.nextid.php*, по функции DB::createSequence() на *http://pear.php.net/manual/en/core.db.createsequence.php* и по функции DB::dropSequence() на *http://pear.php.net/manual/en/core.db.dropsequence.php*.

10.12. Программное создание запросов

Задача

Необходимо создать запрос INSERT или UPDATE из массива, составленного из имен полей. Например, требуется вставить нового пользователя в базу данных. Вместо того чтобы жестко запрограммировать каждое поле информации о пользователе (имя пользователя, почтовый адрес, дата рождения и т. д.), имена полей размещаются в массиве, и на основании этой информации и строится запрос. Так проще осуществлять поддержку, особенно если требуется, в зависимости от обстоятельств, выполнять запросы INSERT или UPDATE с одним и тем же набором полей.

Решение

Для выполнения запроса UPDATE постройте массив из пар поле/значение, а затем объедините все элементы этого массива с помощью функции join():

```
$fields = array('symbol','planet','element');

$update_fields = array();
foreach ($fields as $field) {
    $update_fields[] = "$field = " . $dbh->quote($GLOBALS[$field]);
}
$sql = 'UPDATE zodiac SET ' . join(',',$update_fields)
    . ' WHERE sign = ' . $dbh->quote($sign);
```

Для запроса INSERT создайте массив значений в порядке следования полей и постройте запрос, применяя функцию join() к каждому массиву:

```
$fields = array('symbol','planet','element');

$insert_values = array();
foreach ($fields as $field) {
    $insert_values[] = $dbh->quote($GLOBALS[$field]);
```

```
}
$sql = 'INSERT INTO zodiac (' . join(',',$fields) . ') VALUES ('
       . join(',',$insert_values) . ')';
```

Для PEAR DB версии 1.3 или старше следует применять метод `DB::autoPrepare()`:

```
$fields = array('symbol','planet','element');

// UPDATE: вставьте выражение WHERE
$update_prh = $dbh->autoPrepare('zodiac',$fields,DB_AUTOQUERY_UPDATE,
                                'sign = ?');
$update_values = array();
foreach ($fields as $field) { $update_values[] = $GLOBALS[$field]; }
$update_values[] = $GLOBALS['sign'];
$dbh->execute($update_prh,$update_values);

// INSERT: без выражения WHERE
$insert_prh = $dbh->autoPrepare('zodiac',$fields,DB_AUTOQUERY_INSERT);
$insert_values = array();
foreach ($fields as $field) { $insert_values[] = $GLOBALS[$field]; }
$dbh->execute($insert_prh,$insert_values);
```

Обсуждение

В последних версиях DB метод `DB::autoPrepare()` короткий, и с ним легко работать. PHP 4.2.2 поставляется с версией DB 1.2. Самую свежую версию DB можно загрузить с PEAR. Функция `method_exists()` позволяет проверить, поддерживает ли ваша версия DB функцию `autoPrepare()`:

```
if (method_exists($dbh,'autoPrepare')) {
    $prh = $dbh->autoPrepare('zodiac',$fields,DB_AUTOQUERY_UPDATE',
                             'sign = ?');
    // ...
} else {
    error_log("Can't use autoPrepare");
    exit;
}
```

Если функция `DB::autoPrepare()` недоступна, то можно прибегнуть к показанным в разделе «Решение» приемам обработки массивов, выполняющим ту же самую работу. Если сгенерированная последовательность целых чисел выступает в качестве первичных ключей, то можно объединить способы создания двух запросов в одну функцию. Функция определяет, существует ли запись, а затем генерирует корректный запрос, включая новый идентификатор, как показано в функции `pc_build_query()` примера 10.1.

Пример 10.1. pc_build_query( )

```
function pc_build_query($dbh,$key_field,$fields,$table) {

    if (! empty($_REQUEST[$key_field])) {
        $update_fields = array();
```

```
        foreach ($fields as $field) {
            $update_fields[] = "$field = ".$dbh->quote($_REQUEST[$field]);
        }
        return "UPDATE $table SET " . join(',',$update_fields) .
               " WHERE $key_field = ".$_REQUEST[$key_field];
    } else {
        $insert_values = array();
        foreach ($fields as $field) {
            $insert_values[] = $dbh->quote($_REQUEST[$field]);
        }
        $next_id = $dbh->nextId($table);
        return "INSERT INTO $table ($key_field," . join(',',$fields) .
               ") VALUES ($next_id," . join(',',$insert_values) . ')';
    }
}
```

С помощью этой функции можно сделать простую страничку для редактирования всей информации из таблицы `zodiac`:

```
require 'DB.php';

$dbh = DB::connect('mysql://test:@localhost/test');
$dbh->setFetchMode(DB_FETCHMODE_OBJECT);

$fields = array('sign','symbol','planet','element',
                'start_month','start_day','end_month','end_day');

switch ($_REQUEST['cmd']) {
 case 'edit':
     $row = $dbh->getRow('SELECT ' . join(',',$fields) .
                         " FROM zodiac WHERE id = ?",array($_REQUEST['id']));
 case 'add':
     print '<form method="post" action="'.$_SERVER['PHP_SELF'].'">';
     print '<input type="hidden" name="cmd" value="save">';
     print '<table>';
     if ('edit' == $_REQUEST['cmd']) {
         printf('<input type="hidden" name="id" value="%d">',
                $_REQUEST['id']);
     }
     foreach ($fields as $field) {
         if ('edit' == $_REQUEST['cmd']) {
             $value = htmlspecialchars($row->$field);
         } else {
             $value = '';
         }
         printf('<tr><td>%s: </td><td><input type="text" name="%s" value="%s">,
                $field,$field,$value);
         printf('</td></tr>');
     }
     print '<tr><td></td><td><input type="submit" value="Save"></td></tr>';
     print '</table></form>';
     break;
 case 'save':
```

```
        $sql = pc_build_query($dbh,'id',$fields,'zodiac');
        if (DB::isError($sth = $dbh->query($sql))) {
            print "Couldn't add info: ".$sth->getMessage();
        } else {
            print "Added info.";
        }
        print '<hr>';
    default:
        $sth = $dbh->query('SELECT id,sign FROM zodiac');
        print '<ul>';
        while ($row = $sth->fetchRow()) {
            printf('<li> <a href="%s?cmd=edit&id=%s">%s</a>',
                   $_SERVER['PHP_SELF'],$row->id,$row->sign);
        }
        print '<hr><li> <a href="'.$_SERVER['PHP_SELF'].'?cmd=add">Add New</a>';
        print '</ul>';
        break;
}
```

Оператор `switch` на основе значения элемента `$_REQUEST['cmd']` определяет, какое действие предпримет программа. Если `$_REQUEST['cmd']` равен `add` или `edit`, то программа показывает текстовые окна для каждого поля из массива `$fields`, как показано на рис. 10.1. Если значение элемента `$_REQUEST['cmd']` равно `edit`, то значения для строк с указанным `$id` загружаются из базы данных и отображаются в качестве значений по умолчанию. Если `$_REQUEST['cmd']` равен `save`, то программа вызывает функцию `pc_build_query()`, чтобы сгенерировать соответствующий запрос на вставку или обновление информации в базе данных. После сохранения данных (или если ни один элемент в `$_REQUEST['cmd']`

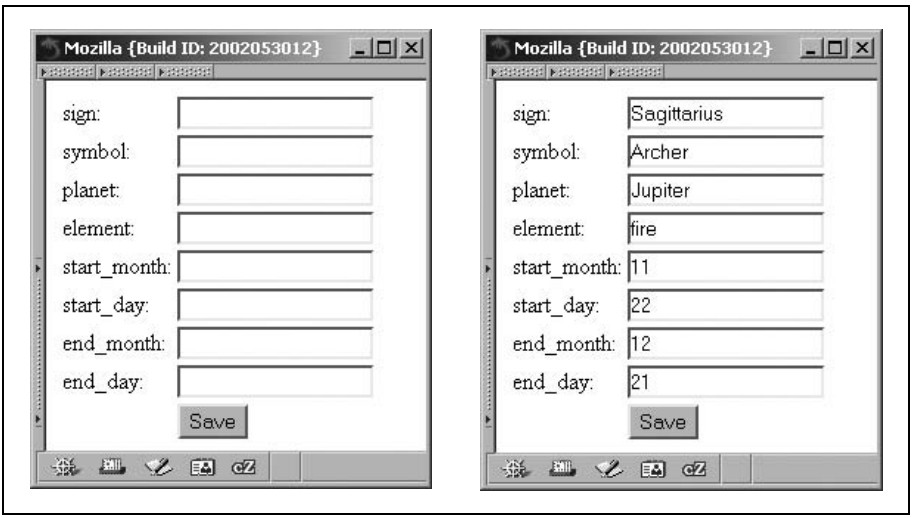

Рис. 10.1. *Добавление и редактирование записи*

не определен), программа выводит список всех знаков зодиака, как показано на рис. 10.2.

Рис. 10.2. *Вывод списка записей*

Определяя, какой запрос, `INSERT` или `UPDATE`, следует строить, функция `pc_build_query()` основывается на присутствии переменной запроса `$_REQUEST['id']` (поскольку `id` передается в переменную `$key_field`). Если переменная `$_REQUEST['id']` не пуста, то эта функция конструирует запрос `UPDATE` для модификации строки с указанным идентификатором. Если переменная `$_REQUEST['id']` пуста (или она вовсе не была установлена), то функция генерирует новый идентификатор с помощью функции `nextId()` и использует этот новый идентификатор в запросе `INSERT`, который добавляет строку в таблицу.

См. также

Документацию по функции `DB::autoPrepare()` на *http://pear.php.net/manual/en/core.db.autoprepare.php*; новую версию PEAR DB, которая доступна на *http://pear.php.net/package-info.php?package=DB*.

10.13. Постраничный вывод большого количества записей

Задача

Необходимо отобразить на странице большой набор данных и обеспечить ссылки, позволяющие перемещаться по этому набору данных.

Решение

Здесь нужен PEAR класс DB_Pager:

```
require 'DB/Pager.php';

$offset = intval($_REQUEST['offset']);
$per_page = 3;

$sth = $dbh->query('SELECT * FROM zodiac ORDER BY id');
$pager = new DB_Pager($sth, $offset, $per_page);
$data = $pager->build();

// выводим на эту страницу каждую строку
while ($v = $pager->fetchRow()) {
    print "$v->sign, $v->symbol ($v->id)<br>";
}

// ссылка на предыдущую страницу
printf('<a href="%s?offset=%d">&lt;&lt;Prev</a> |',
       $_SERVER['PHP_SELF'],$data['prev']);

// прямая ссылка на каждую страницу
foreach ($data['pages'] as $page => $start) {
    printf(' <a href="%s?offset=%d">%d</a>
|',$_SERVER['PHP_SELF'],$start,$page);
}

// ссылка на следующую страницу
printf(' <a href="%s?offset=%d">Next&gt;&gt;</a>',
           $_SERVER['PHP_SELF'],$data['next']);

// показываем, какие записи находятся на данной странице
printf("<br>(Displaying %d - %d of %d)",
       $data['from'],$data['to'],$data['numrows']);
```

Если класса DB_Pager нет или вы не хотите его использовать, то можно реализовать свое собственное отображение индексированной ссылки с помощью функций pc_indexed_links() и pc_print_link(), показанных в разделе «Обсуждение» в примерах 10.2 и 10.3.

```
$offset = intval($_REQUEST['offset']);
if (! $offset) { $offset = 1; }
$per_page = 5;
$total = $dbh->getOne('SELECT COUNT(*) FROM zodiac');

$sql = $dbh->modifyLimitQuery('SELECT * FROM zodiac ORDER BY id',
                              $offset - 1,$per_page);
$ar = $dbh->getAll($sql);
foreach ($ar as $k => $v) {
    print "$v->sign, $v->symbol ($v->id)<br>";
}

pc_indexed_links($total,$offset,$per_page);
printf("<br>(Displaying %d - %d of %d)",$offset,$offset+$k,$total);
```

Обсуждение

Класс DB_Pager разработан специально для постраничного отображения результатов запроса PEAR DB. Для того чтобы его использовать, создайте объект класса DB_Pager и укажите ему, какой запрос послать, какое смещение задать в начале результирующего множества, и какое количество элементов должно находиться на каждой странице. Это обеспечит корректное размещение информации по страницам.

Метод $pager->build() определяет возвращаемые строки и другие переменные, необходимые для конкретной страницы. Класс DB_Pager предоставляет метод fetchRow() для извлечения результатов, который работает таким же образом, как метод класса DB. (В классе DB_Pager можно также использовать метод fetchInto()). Однако, хоть он и предоставляет всю необходимую информацию для построения соответствующих ссылок, но оставляет вам собственно разработку этих ссылок. Начальное смещение предыдущей страницы находится в переменной $data['prev'], а переменная $data['next'] содержит начальное смещение следующей страницы. Массив $data['pages'] хранит номера страниц и их начальные смещения. Вывод для случая, когда переменная $offset равна 0, показан на рис. 10.3.

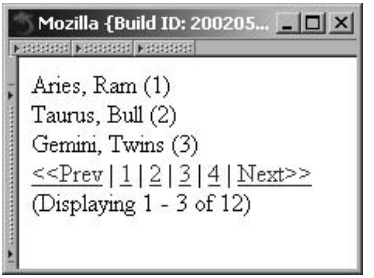

Рис. 10.3. *Постраничный вывод с использованием класса DB_Pager*

Все номера страниц, а также «<<Prev» и «Next>>» представляют собой ссылки. «<<Prev» и «1» указывают на текущую страницу; другие ссылки указывают на соответствующие страницы. На странице 4 ссылка «Next>>» указывает обратно на страницу 1. (Но ссылка «<<Prev» на странице 1 не указывает на страницу 4.) Номера в ссылках относятся к номерам страниц, а не к номерам элементов.

Если класс DB_Pager недоступен, то получить соответствующим образом отформатированные ссылки позволяют функции pc_print_link() и pc_indexed_links(), показанные в примерах 10.2 и 10.3.

Пример 10.2. pc_print_link( )

```
function pc_print_link($inactive,$text,$offset='') {
    if ($inactive) {
```

```
        printf('<font color="#666666">%s</font>',$text);
    } else {
        printf('<a href="%s?offset=%d">%s</
a>',$_SERVER['PHP_SELF'],$offset,$text);
    }
}
```

Пример 10.3. pc_indexed_links( )

```
function pc_indexed_links($total,$offset,$per_page) {
    $separator = ' | ';

    // выводим ссылку "<<Prev"
    pc_print_link($offset == 1, '&lt;&lt;Prev', $offset - $per_page);


    // выводим все группировки, за исключением последней
    for ($start = 1, $end = $per_page;
         $end < $total;
         $start += $per_page, $end += $per_page) {

        print $separator;
        pc_print_link($offset == $start, "$start-$end", $start);
    }

    /* выводим последнюю группировку - в этой точке переменная $start
     * указывает на элемент в начале последней группировки
     */

    /* текст толжен содержать диапазон, только если на последней
     * странице находится более одного элемента. Например,
     * последняя группировка из 11 элементов по 5 на каждой
     * странице должна показать просто "11", а не "11-11"
     */
    $end = ($total > $start) ? "-$total" : '';

    print $separator;
    pc_print_link($offset == $start, "$start$end", $start);

    // выводим ссылку "Next>>"
    print $separator;
    pc_print_link($offset == $start, 'Next&gt;&gt;',$offset + $per_page);
}
```

Применяя эти функции, извлеките соответствующее подмножество данных с помощью метода `DB::modifyLimitQuery()` **и выведите их. Для отображения индексированных ссылок вызовите функцию** `pc_indexed_links()`**:**

```
$offset = intval($_REQUEST['offset']);
if (! $offset) { $offset = 1; }
$per_page = 5;
$total = $dbh->getOne('SELECT COUNT(*) FROM zodiac');

$sql = $dbh->modifyLimitQuery('SELECT * FROM zodiac ORDER BY id',
                              $offset - 1,$per_page);
$ar = $dbh->getAll($sql);
```

```
foreach ($ar as $k => $v) {
    print "$v->sign, $v->symbol ($v->id)<br>";
}

pc_indexed_links($total,$offset,$per_page);
printf("<br>(Displaying %d - %d of %d)",$offset,$offset+$k,$total);
```

После соединения с базой данных необходимо убедиться, что переменная `$offset` имеет соответствующее значение. Переменная `$offset` представляет начальную запись в результирующем множестве, которое требуется отобразить. Чтобы стартовать с начала результирующего набора данных, надо установить переменную `$offset` в 1. В переменную `$per_page` заносится количество записей, отображаемых на каждой странице, а переменная `$total` представляет общее количество записей во всем результирующем множестве. В данном примере показываются все записи из таблицы Zodiac, поэтому переменной `$total` присваивается количество всех строк в таблице.

SQL-запрос, извлекающий данные в соответствующем порядке, имеет вид:

```
SELECT * FROM zodiac ORDER BY id
```

Для ограничения извлекаемых строк вызовите функцию `modifyLimitQuery()`. Чтобы извлечь `$per_page` строк, надо начинать с `$offset - 1`, поскольку первая строка в базе данных имеет номер 0, а не 1. Для ограничения строк, возвращаемых запросом, метод `modifyLimitQuery()` реализует корректный алгоритм, специфический для конкретной базы данных.

Выбранные строки извлекаются с помощью метода `$dbh->getAll($sql)`, а затем отображается информация, содержащаяся в каждой строке. В конце каждой строки функция предоставляет навигационные ссылки. Вывод для случая, когда переменная `$offset` не установлена (или равна 1), показан на рис. 10.4.

Рис. 10.4. *Постраничный вывод результатов, полученный с помощью функции pc_indexed_links( )*

На рис. 10.4 «6-10», «11-12» и «Next>>» представляют собой ссылки на одну и ту же страницу с заданными аргументами `$offset`, тогда как «<<Prev» и «1-5» не активны, поскольку то, на что они ссылаются, в настоящее время отображается.

См. также

Информацию по классу `DB_Pager` на *http://pear.php.net/package-info.php?package=DB_Pager*.

10.14. Кэширование запросов и результатов

Задача

Требуется исключить повторный запуск потенциально ресурсоемких запросов в базу данных, если их результаты не изменялись.

Решение

Используйте пакет PEAR `Cache_DB`. Он предоставляет в качестве оболочки уровня абстракции базы данных `DB` объект, который имеет схожие методы и автоматически кэширует результаты запросов `SELECT`:

```
require 'Cache/DB.php';

$cache = new Cache_DB;
$cache->connect('mysql://test:@localhost/test');

$sth = $cache->query("SELECT sign FROM zodiac WHERE element LIKE 'fire'");

while($row = $sth->fetchRow()) {
    print $row['sign'].'\n';
}
```

Обсуждение

Пакет `Cache_DB` используется почти так же, как и `DB`, но есть некоторые существенные отличия. Во-первых, требуется файл *Cache/DB.php* вместо *DB.php*. В этом случае файл *Cache/DB.php* загружает соответствующие классы DB. Вместо того чтобы посредством метода `DB::connect()` создавать спецификатор базы данных, с помощью оператора `new` создается объект `Cache_DB`, а затем вызывается метод `connect()` этого объекта. Синтаксис метода `$cache->connect()` тот же самый, однако ему достаточно передать DSN, идентифицирующий базу данных. Метод `query()` пакета `Cache_DB` работает точно так же, как и в `DB`, однако в `Cache_DB` нет методов `prepare()` и `execute()`. Метод `query()` возвращает спецификатор оператора, поддерживающего методы `fetchRow()` и `fetchInto()`, но режим выборки по умолчанию определяется константой `DB_FETCH_ASSOC`, а не `DB_FETCH_ORDERED`.

Когда какой-либо оператор SELECT первый раз передается методу $cache->query(), то Cache_DB выполняет оператор и возвращает результаты точно так же, как и DB, но помимо этого он записывает результаты в файл, имя которого представляет собой хеш запроса. Если тот же самый оператор SELECT снова передается методу $cache->query(), то вместо того чтобы запрашивать базу данных, Cache_DB извлекает результаты из файла.

По умолчанию Cache_DB создает свои кэш-файлы в подкаталоге текущего каталога с именем *db_query*. Его можно изменить, передав имя каталога в составе массива параметров как второй аргумент конструктору Cache_DB. Следующий оператор определяет кэш-каталог как */tmp/db_query*:

```
$cache = new Cache_DB('file',array('cache_dir' => '/tmp/'));
```

Первый аргумент, file, указывает Cache_DB, какой контейнер использовать для хранения кэшированных данных. По умолчанию это значение file, но при этом необходимо указать еще и параметры контейнера во втором аргументе. Подходящим является cache_dir, указывающий Cache_DB, где создавать подкаталог *db_query*. Символ косой черты в конце обязателен.

По умолчанию информация в кэше хранится в течение часа. Этим временем можно управлять, передавая другое значение (в секундах) при создании нового объекта Cache_DB. Ниже показано, как сохранить информацию в кэше в течение одного дня, или 86 400 секунд:

```
$cache = new Cache_DB('file',array('cache_dir' => '.',
                                   'filename_prefix' => 'query_'),86400);
```

Время действия передается в третьем аргументе, поэтому необходимо также передать значения по умолчанию для первых двух аргументов.

Содержимое кэша остается прежним, даже если база данных изменяется в результате запросов INSERT, UPDATE, или DELETE. Если в кэше хранится результат оператора SELECT, относящийся к данным, которых уже не существует в базе данных, то необходимо полностью очистить кэш непосредственно с помощью метода $cache->flush():

```
$cache->flush('db_cache');
```

Очень важно включить аргумент db_cache в вызов flush(). Кэш-система в PEAR поддерживает разделение кэшированных данных на различные группы, а объект Cache_DB помещает всю информацию, за которой он следит, в группу db_cache. Пропуск аргумента группы приведет к удалению файлов из базового каталога кэша (из которого, возможно, запускается ваш сценарий).

Файловый контейнер хранит каждый результат в файле с именем, основанным на MD5-хеше запроса, сгенерировавшего определенный результат. Поскольку MD5 чувствителен к регистру, то и файловый контейнер также чувствителен к регистру. Это означает, что если резуль-

таты запроса SELECT * FROM zodiac находятся в кэше, а запускается запрос SELECT * from zodiac, то его результаты не будут найдены в кэше, и он будет запущен как новый запрос. Последовательное применение заглавных букв, пробелов и порядка полей при создании запроса приводит к более эффективному использованию кэша.

Этот рецепт посвящен, в основном, файловому контейнеру, но кэш-система в PEAR поддерживает множество других контейнеров, хранящих кэшированные данные, например, разделяемую память, PHPLib-сеансы, базы данных на основе библиотеки dbx и сеансы msession. Для того чтобы использовать другой контейнер, надо при создании нового объекта Cache_DB передать соответствующее имя контейнера в качестве первого аргумента:

```
$cache = new Cache_DB('shm');
```

См. также

Информацию о кэш-системе PEAR и различных контейнерах на *http://pear.php.net/package-info.php?package=Cache*.

10.15. Программа: Хранение сообщений форума, разбитых на темы

Сохранение и извлечение сообщений, относящихся к различным темам (разделенных на потоки), требует особой осторожности при отображении тем в определенном порядке. Определение потомка каждого сообщения и построение дерева отношений сообщений может привести к рекурсии запросов. Пользователи в основном просматривают список сообщений и читают отдельные сообщения значительно чаще, чем помещают свои собственные сообщения. Потратив небольшие дополнительные усилия при записи нового сообщения в базу данных, можно упростить запрос, извлекающий список показываемых сообщений, и сделать его значительно более эффективным.

Сохраним сообщения в таблице, имеющей, например, такую структуру:

```
CREATE TABLE pc_message (
  id INT UNSIGNED NOT NULL,
  posted_on DATETIME NOT NULL,
  author CHAR(255),
  subject CHAR(255),
  body MEDIUMTEXT,
  thread_id INT UNSIGNED NOT NULL,
  parent_id INT UNSIGNED NOT NULL,
  level INT UNSIGNED NOT NULL,
  thread_pos INT UNSIGNED NOT NULL,
  PRIMARY KEY(id)
);
```

Первичный ключ id – это уникальное целочисленное значение, идентифицирующее конкретное сообщение. Время и дата отправки сообщения хранится в поле posted_on, а поля author (автор), subject (тема) и body (содержимое) представляют (кто бы мог подумать!) автора, тему и содержимое сообщения. Остальные четыре поля отслеживают связи между сообщениями в потоке. Целочисленное значение thread_id определяет каждый поток. Все сообщения в определенном потоке имеют одинаковое значение поля thread_id. Если сообщение является ответом на другое сообщение, то поле parent_id представляет идентификатор сообщения, на которое отвечают. Поле level показывает уровень вложенности ответа на сообщение в потоке. Первое сообщение в потоке имеет уровень 0. Ответ на это сообщение нулевого уровня имеет уровень 1, а ответ на ответное сообщение уровня 1 имеет уровень 2. Несколько сообщений в потоке могут иметь одинаковые значения поля level и одинаковые значение поля parent_id. Например, если кто-то начинает поток сообщений о преимуществах операционной системы BeOS перед CP/M, все сердитые отклики на это сообщение от многочисленных приверженцев системы CP/M имеют уровень 1 и значение поля parent_id, равное идентификатору исходного сообщения.

Именно последнее поле, thread_pos, собственно, и позволяет упростить показ сообщения. Все сообщения потока при отображении упорядочиваются по значению их поля thread_pos.

Ниже приведены правила вычисления значения поля thread_pos:

- Первое сообщение в потоке имеет thread_pos = 0.
- Для нового сообщения N, при условии отсутствия в потоке сообщений, имеющих того же родителя, что и у N, значение поля thread_pos на единицу больше значения поля thread_pos его родителя.
- Для нового сообщения N, если в потоке есть сообщения с тем же родителем, что и у сообщения N, значение поля thread_pos на единицу больше, чем самое большое значение thread_pos у сообщений с тем же родителем.
- После того как определено значение поля thread_pos нового сообщения, все сообщения того же потока со значением поля thread_pos, большим или равным значению поля сообщения N, получают значение поля thread_pos, увеличенное на 1 (чтобы освободить пространство для сообщения N).

Программа форума, *message.php*, показанная в примере 10.4, сохраняет сообщения и вычисляет значения поля. Простой вывод показан на рис. 10.5.

Пример 10.4. message.php

```
require 'DB.php';

// полезная функция для отладки базы данных
function log_die($ob) { print '<pre>'; print_r($ob); print '</pre>'; }
```

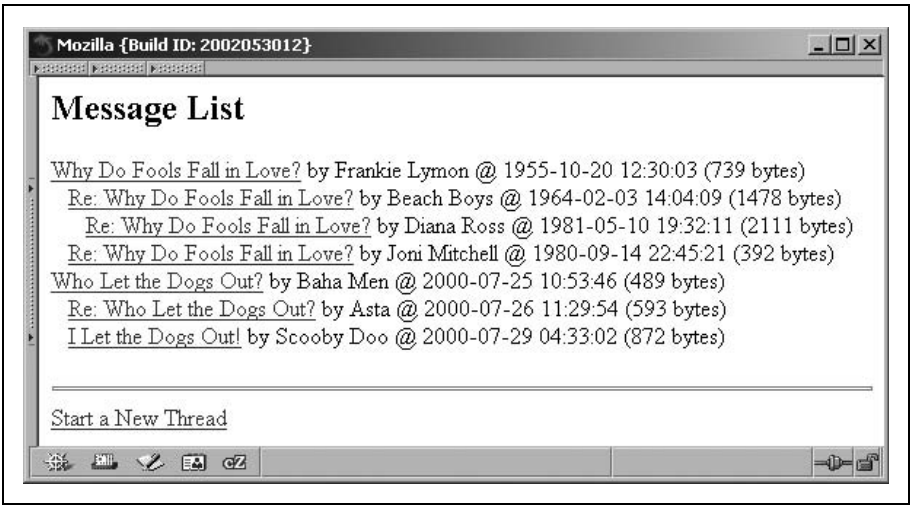

Рис. 10.5. *Доска сообщений, разделенных на потоки*

```
// соединяемся с базой данных
$dbh = DB::connect('mysql://test:@localhost/test') or die("Can't connect");
if (DB::isError($dbh)) { log_die($dbh); }
$dbh->setFetchMode(DB_FETCHMODE_OBJECT);
PEAR::setErrorHandling(PEAR_ERROR_CALLBACK,'log_die');

// Значение $_REQUEST['cmd'] говорит нам, что делать
switch ($_REQUEST['cmd']) {
case 'read':                    // читаем отдельное сообщение
     pc_message_read();
     break;
case 'post':                    // отображаем форму для посылки сообщения
     pc_message_post();
     break;
case 'save':                        // записываем посланное сообщение
     if (pc_message_validate()) { // если сообщение допустимое,
         pc_message_save();       // то сохраняем его
         pc_message_list();       // и выводим список сообщений
     } else {
         pc_message_post();       // в противном случае, снова выводим
                                     форму для сообщения
     }
     break;
case 'list':                    // выводим список сообщений по умолчанию
default:
     pc_message_list();
     break;
}

// функция pc_message_save() записывает сообщение в базу данных
function pc_message_save() {
```

```
global $dbh;

$parent_id = intval($_REQUEST['parent_id']);

/* синтаксис MySQL, гарантирующий, что pc_message не изменяет
 * значение, с которым мы работаем. Необходимо также заблокировать
 * таблицы, которые содержат поток и последовательности pc_message
 */
$dbh->query('LOCK TABLES pc_message WRITE, thread_seq WRITE,
             pc_message_seq WRITE');

// является ли сообщение ответом?
if ($parent_id) {

    // получаем поток, уровень и  thread_pos родительского сообщения
    $parent = $dbh->getRow("SELECT thread_id,level,thread_pos
                            FROM pc_message
                            WHERE id = $parent_id");

    // уровень ответа на единицу больше, чем у его родителя
    $level = $parent->level + 1;

    /* каково максимальное значение thread_pos среди сообщений потока
       с тем же самым родителем? */
    $thread_pos = $dbh->getOne("SELECT MAX(thread_pos) FROM pc_message
       WHERE thread_id = $parent->thread_id AND parent_id = $parent_id");

    // существуют ли ответы для данного родителя?
    if ($thread_pos) {
        // это thread_pos следует сразу за наибольшим
        $thread_pos++;
    } else {
        // это первый ответ, поэтому помещаем его сразу после родителя
        $thread_pos = $parent->thread_pos + 1;
    }

    /* увеличиваем значение thread_pos всех сообщений потока, которые
       идут вслед за этим сообщением */
    $dbh->query("UPDATE pc_message SET thread_pos = thread_pos + 1
        WHERE thread_id = $parent->thread_id AND thread_pos >=
                           $thread_pos");

    // новое сообщение должно быть записано с родительским thread_id
    $thread_id = $parent->thread_id;
} else {
    // сообщение не является ответом, поэтому оно открывает новый поток
    $thread_id = $dbh->nextId('thread');
    $level = 0;
    $thread_pos = 0;
}

// получаем новый идентификатор для этого сообщения
$id = $dbh->nextId('pc_message');

/* вставляем сообщение в базу данных.С помощью
   функций prepare() и execute() обеспечиваем соответствующее
```

```
        заключение всех полей в кавычки */
    $prh =
        $dbh->prepare("INSERT INTO pc_message (id,thread_id,parent_id,
                       thread_pos,posted_on,level,author,subject,body)
                       VALUES (?,?,?,?,NOW(),?,?,?,?)");

    $dbh->execute($prh,array($id,$thread_id,$parent_id,$thread_pos,$level,
                             $_REQUEST['author'],$_REQUEST['subject'],
                             $_REQUEST['body']));


    // Сообщаем MySQL, что остальные могут теперь использовать
    // таблицу pc_message
    $dbh->query('UNLOCK TABLES');
}

// функция pc_message_list() выводит список всех сообщений
function pc_message_list() {
    global $dbh;

    print '<h2>Message List</h2><p>';

    /* упорядочиваем сообщения в соответствии с их потоком (thread_id)
       и их позицией внутри потока (thread_pos) */
    $sth = $dbh->query("SELECT id,author,subject,LENGTH(body) AS body_length,
                        posted_on,level FROM pc_message
                        ORDER BY thread_id,thread_pos");
    while ($row = $sth->fetchRow()) {
        // делаем отступ для сообщений с уровнем > 0
        print str_repeat(' ',4 * $row->level);
        // выводим информацию о сообщении со ссылкой для его чтения
        print<<<_HTML_
<a href="$_SERVER[PHP_SELF]?cmd=read&id=$row->id">$row->subject</a> by
$row->author @ $row->posted_on ($row->body_length bytes)
<br>
_HTML_;
    }

    // предоставляем возможность послать сообщение, не являющееся ответом
    printf('<hr><a href="%s?cmd=post">Start a New Thread</a>',
           $_SERVER['PHP_SELF']);
}

// функция pc_message_read() выводит отдельное сообщение
function pc_message_read() {
    global $dbh;

    /* проверяем, что идентификатор переданного нами сообщения является
       целым числом и действительно представляет сообщение */
    $id = intval($_REQUEST['id']) or die("Bad message id");
    if (! ($msg = $dbh->getRow(
        "SELECT author,subject,body,posted_on FROM pc_message
                                              WHERE id = $id"))) {
        die("Bad message id");
    }
```

```
    /* не выводим введенный пользователем HTML-текст, но отображаем
       символ новой строки как HTML-ограничитель строки */
    $body = nl2br(strip_tags($msg->body));

    // выводим сообщение со ссылками на ответ и возвращаем список сообщений
    print<<<_HTML_
<h2>$msg->subject</h2>
<h3>by $msg->author</h3>
<p>
$body
<hr>
<a href="$_SERVER[PHP_SELF]?cmd=post&parent_id=$id">Reply</a>
<br>
<a href="$_SERVER[PHP_SELF]?cmd=list">List Messages</a>
_HTML_;
}

// функция pc_message_post() выводит форму для посылаемого сообщения
function pc_message_post() {
    global $dbh,$form_errors;

    foreach (array('author','subject','body') as $field) {
        // преобразует символы значений полей по умолчанию
        // в escape-последовательности
        $$field = htmlspecialchars($_REQUEST[$field]);
        // окрашивает сообщения об ошибках в красный цвет
        if ($form_errors[$field]) {
            $form_errors[$field] = '<font color="red">' .
                $form_errors[$field] . '</font><br>';
        }
    }

    // если это сообщение является ответом
    if ($parent_id = intval($_REQUEST['parent_id'])) {

        // вместе с представлением формы посылаем parent_id
        $parent_field =
            sprintf('<input type="hidden" name="parent_id" value="%d">',
                    $parent_id);

        // если тему сообщения не передали, используем родительскую тему
        if (! $subject) {
            $parent_subject = $dbh->getOne('SELECT subject FROM pc_message
                                            WHERE id = ?',array($parent_id));
            /* префикс 'Re: ' к родительской теме, если она существует,
               но еще не имеет префикса 'Re:' */
            $subject = htmlspecialchars($parent_subject);
           if ($parent_subject && (! preg_match('/^re:/i',$parent_subject)))
{
                $subject = "Re: $subject";
            }
        }
    }
```

```
    // выводим форму отправки сообщения с ошибками и значениями по умолчанию
    print<<<_HTML_
<form method="post" action="$_SERVER[PHP_SELF]">
<table>
<tr>
 <td>Your Name:</td>
 <td>$form_errors[author]<input type="text" name="author" value="$author">
</td>
<tr>
 <td>Subject:</td>
 <td>$form_errors[subject]<input type="text" name="subject" value="$subject">
</td>
<tr>
 <td>Message:</td>
 <td>$form_errors[body]<textarea rows="4" cols="30" wrap="physical"
name="body">$body</textarea>
</td>
<tr><td colspan="2"><input type="submit" value="Post Message"></td></tr>
</table>
$parent_field
<input type="hidden" name="cmd" value="save">
</form>

_HTML_;
}

// функция pc_message_validate() обеспечивает
// наличие какого-либо ввода в каждом поле
function pc_message_validate() {
    global $form_errors;

    $form_errors = array();

    if (! $_REQUEST['author']) {
        $form_errors['author'] = 'Please enter your name.';
    }
    if (! $_REQUEST['subject']) {
        $form_errors['subject'] = 'Please enter a message subject.';
    }
    if (! $_REQUEST['body']) {
        $form_errors['body'] = 'Please enter a message body.';
    }

    if (count($form_errors)) {
        return false;
    } else {
        return true;
    }
}
```

Для корректной реализации совместного использования функции `pc_message_save()` необходим монопольный доступ к таблице `msg` в промежутке времени между началом вычисления значения поля `thread_ pos`

нового сообщения и моментом действительной записи нового сообщения в базу данных. Чтобы обеспечить это, мы воспользовались командами MySQL's `LOCK TABLE` и `UNLOCK TABLES`. В других базах данных синтаксис может отличаться, а может понадобиться стартовать транзакцию в начале функции и фиксировать ее в конце.

Во время вывода сообщений можно использовать поле `level` для ограничения извлекаемой из базы данных информации. Значительное увеличение глубины вложенности потоков обсуждения может предотвратить чрезмерное разрастание страниц. Например, ниже показано, как отобразить только первое сообщение каждого потока и все ответы на это первое сообщение:

```
$sth = $dbh->query(
    "SELECT * FROM msg WHERE level <= 1 ORDER BY thread_id,thread_pos");
while ($row = $sth->fetchRow()) {
    // выводим каждое собщение
}
```

Для создания группы обсуждения на веб-сайте можно воспользоваться существующими PHP-пакетами для форумов. Наиболее популярным является Phorum (*http://www.phorum.org/*), а список множества других пакетов находится на *http://www.zend.com/apps.php?CID=261*.

11

Автоматизация работы с Web

11.0. Введение

Большую часть времени PHP работает как часть веб-сервера, посылая информацию броузерам. Даже если его запускают из командной строки, он, как правило, выполняет задачу и выводит некоторую информацию. Тем не менее PHP может быть также полезен и в качестве веб-броузера – получая доступ к определенным URL и обрабатывая их содержимое. Большинство рецептов этой главы посвящено получению доступа к URL и обработке результатов, хотя здесь обсуждаются и некоторые другие задачи, такие как использование шаблонов и обработка серверных протоколов.

В PHP есть четыре способа доступа к удаленным URL. Выбор зависит от того, насколько доступ должен быть простым, управляемым и переносимым. Эти четыре способа реализуются посредством функций `fopen()` и `fsockopen()`, расширения cURL или класса `HTTP_Request` в PEAR.

С функцией `fopen()` работать просто и удобно. Она рассматривается в рецепте 11.1. Функция `fopen()` поддерживает перенаправления, поэтому если она применяется для получения доступа к каталогу *http://www.example.com/people*, а сервер переадресует вас к *http://www.example.com/people/*, то в результате будет получена страница с индексом каталогов, а не сообщение о том, что URL был перемещен. Функция `fopen()` также работает и с HTTP и с FTP. Оборотная сторона функции `fopen()` заключается в том, что она может обрабатывать только HTTP-запросы GET (не HEAD или POST), вместе с запросом невозможно посылать дополнительные заголовки или любые cookies, и получить можно только содержимое ответа, но не его заголовки.

Работа с функцией `fsockopen()` требует больше затрат, но предоставляет большую гибкость. Функция `fsockopen( )` показана в рецепте 11.2. После открытия сокета с помощью функции `fsockopen()` необходимо послать соответствующий HTTP-запрос на данный сокет, а затем прочитать и проанализировать ответ. Это позволяет добавлять заголовки

к запросу и предоставляет доступ ко всем заголовкам ответа. Однако при этом надо написать дополнительный код, для того чтобы правильно проанализировать ответ и предпринять соответствующее действие, например, последовать переадресации.

Если у вас есть доступ к расширению cURL или к классу `HTTP_Request` в PEAR, то следует применять эти инструменты, а не функцию `fsockopen()`. Расширение cURL поддерживает множество различных протоколов (включая HTTPS, рассмотренный в рецепте 11.5) и предоставляет доступ к заголовкам ответа. Мы используем расширение cURL в большинстве рецептов этой главы. Для того чтобы с этим расширением можно было работать, необходимо инсталлировать библиотеку cURL, доступную на *http://curl.haxx.se*. Кроме того, PHP должен быть собран с ключом `--with-curl`.

Класс PEAR `HTTP_Request`, фигурирующий в рецептах 11.2, 11.3 и 11.4, не поддерживает HTTPS, но предоставляет доступ к заголовкам и может использовать любой метод HTTP. Если этот модуль PEAR не установлен, то его можно загрузить с *http://pear.php.net/get/HTTP_Request*. Как только файлы модуля окажутся в каталоге `include_path`, с ними можно будет работать, обеспечивая хорошую переносимость решения.

Рецепт 11.6 помогает проникнуть за кулисы HTTP-запроса, чтобы исследовать заголовки запроса и ответа. Если запрос, конструируемый в программе, не дает желаемого результата, то изучение заголовков часто дает ключ к разгадке причины ошибки.

Рецепты с 11.7 по 11.11 помогут организовать обработку содержимого веб-страницы, загруженной в программу. В первом из них показано, как отмечать определенные слова на странице цветными блоками. Этот способ полезен, например, для подсвечивания условий поиска. Рецепт 11.8 содержит функцию для нахождения всех ссылок на странице. Это важный строительный блок для создания веб-паука (web spider) или системы контроля ссылок. Преобразованию простого ASCII-текста в HTML посвящены рецепты 11.9 и 11.10. Рецепт 11.11 показывает, как удалить все HTML- и PHP-теги из веб-страницы.

Другой вид работы со страницами состоит в применении системы шаблонов. Шаблоны, а они рассматриваются в рецепте 11.12, позволяют как угодно изменять внешний вид ваших веб-страниц, не меняя кода PHP, заполняющего страницы динамическими данными. Точно так же можно изменять код программы управления страницами, не затрагивая внешний вид последних. Рецепт 11.13 посвящен общей задаче администрирования сервера – анализу файлов протоколов доступа к веб-сайту.

Для примера две программы используют инструмент извлечения ссылок, описанный в рецепте 11.8. Программа в рецепте 11.14 просматривает ссылки на странице и сообщает, какие ссылки еще действительны, какие были перемещены, а какие больше не работают. Программа

в рецепте 11.15 докладывает о состоянии ссылок. Она сообщает, перемещалась ли страница, на которую ссылаются, и когда она была изменена последний раз.

11.1. Получение содержимого URL методом GET

Задача

Необходимо получить содержимое URL. Например, требуется вставить часть одной веб-страницы в содержимое другой страницы.

Решение

Передайте URL функции `fopen()` и получите содержимое страницы с помощью функции `fread()`:

```
$page = '';
$fh = fopen('http://www.example.com/robots.txt','r') or die($php_errormsg);
while (! feof($fh)) {
    $page .= fread($fh,1048576);
}
fclose($fh);
```

Можно прибегнуть к расширению cURL:

```
$c = curl_init('http://www.example.com/robots.txt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
$page = curl_exec($c);
curl_close($c);
```

А можно применить класс `HTTP_Request` из PEAR:

```
require 'HTTP/Request.php';

$r = new HTTP_Request('http://www.example.com/robots.txt');
$r->sendRequest();
$page = $r->getResponseBody();
```

Обсуждение

Для того чтобы получить доступ к защищенной странице, можно поместить имя пользователя и пароль в URL. В приведенном ниже примере имя пользователя – `david`, а пароль – `hax0r`. Покажем, как это сделать с помощью функции `fopen()`:

```
$fh = fopen('http://david:hax0r@www.example.com/secrets.html','r')
    or die($php_errormsg);
while (! feof($fh)) {
    $page .= fread($fh,1048576);
}
fclose($fh);
```

Ниже показано, как это выполнить, применяя расширение cURL:

```
$c = curl_init('http://www.example.com/secrets.html');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_USERPWD, 'david:hax0r');
$page = curl_exec($c);
curl_close($c);
```

Теперь посмотрим, как это сделать с помощью класса HTTP_Request:

```
$r = new HTTP_Request('http://www.example.com/secrets.html');
$r->setBasicAuth('david','hax0r');
$r->sendRequest();
$page = $r->getResponseBody();
```

Функция fopen() следует переадресациям, определенным в заголовках ответа Location, а класс HTTP_Request этого не делает. Расширение cURL подчиняется им, только если установлен параметр CURLOPT_FOLLOWLOCATION:

```
$c = curl_init('http://www.example.com/directory');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_FOLLOWLOCATION, 1);
$page = curl_exec($c);
curl_close($c);
```

Расширение cURL может делать некоторые другие вещи с извлеченной страницей. Если установлен параметр CURLOPT_RETURNTRANSFER, то функция curl_exec() возвращает строку, содержащую полученную страницу:

```
$c = curl_init('http://www.example.com/files.html');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
$page = curl_exec($c);
curl_close($c);
```

Для записи извлеченной страницы в файл откройте дескриптор файла на запись с помощью функции fopen() и установите для этого дескриптора файла параметр CURLOPT_FILE:

```
$fh = fopen('local-copy-of-files.html','w') or die($php_errormsg);
$c = curl_init('http://www.example.com/files.html');
curl_setopt($c, CURLOPT_FILE, $fh);
curl_exec($c);
curl_close($c);
```

Чтобы передать ресурсы cURL и содержимое полученной страницы в функцию, установите значение параметра CURLOPT_WRITEFUNCTION равным имени этой функции:

```
// сохраняем URL и содержимое страницы в базу данных
function save_page($c,$page) {
    $info = curl_getinfo($c);
    mysql_query("INSERT INTO pages (url,page) VALUES ('" .
                mysql_escape_string($info['url']) . "', '" .
                mysql_escape_string($page) . "')");
```

```
}

$c = curl_init('http://www.example.com/files.html');
curl_setopt($c, CURLOPT_WRITEFUNCTION, 'save_page');
curl_exec($c);
curl_close($c);
```

Если ни один из параметров CURLOPT_RETURNTRANSFER, CURLOPT_FILE или CURLOPT_WRITEFUNCTION не установлен, то расширение cURL выводит содержимое возвращенной страницы.

Функция fopen() вместе с параметрами include и require может получать доступ к удаленным файлам, только если доступ к таковым разрешен. А по умолчанию он разрешен и управляется с помощью параметра настройки allow_url_fopen. Однако в Windows опции include и require не дают возможности извлекать удаленные файлы в версиях PHP более ранних, чем 4.3, даже если параметр allow_url_fopen установлен в on.

См. также

Рецепт 11.2 о получении содержимого URL с помощью метода POST; рецепт 18.3, в котором рассматривается открытие удаленных файлов с помощью функции fopen(); документацию по функции fopen() на *http://www.php.net/fopen*, по функции include на *http://www.php.net/include*, по функции curl_init() на *http://www.php.net/curl-init*, по функции curl_setopt() на *http://www.php.net/curl-setopt*, по функции curl_exec() на *http://www.php.net/curl-exec* и по функции curl_close() на *http://www.php.net/curl-close*; о классе PEAR HTTP_Request на *http://pear.php.net/package-info.php?package=HTTP_Request*.

11.2. Извлечение содержимого URL с помощью метода POST

Задача

Необходимо получить содержимое URL с помощью метода POST, а не метода GET, применяемого обычно. Например, требуется отправить HTML-форму.

Решение

Это делается при помощи расширения cURL с установленным параметром CURLOPT_POST:

```
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_POST, 1);
curl_setopt($c, CURLOPT_POSTFIELDS, 'monkey=uncle&rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
$page = curl_exec($c);
curl_close($c);
```

Если расширение cURL недоступно, то применяется класс `HTTP_Request` из PEAR:

```
require 'HTTP/Request.php';

$r = new HTTP_Request('http://www.example.com/submit.php');
$r->setMethod(HTTP_REQUEST_METHOD_POST);
$r->addPostData('monkey','uncle');
$r->addPostData('rhino','aunt');
$r->sendRequest();
$page = $r->getResponseBody();
```

Обсуждение

Посылка запроса с методом POST требует специальной обработки любых аргументов. В запросе GET эти аргументы заключены в строке запроса, но в запросе POST они находятся в теле запроса. Кроме того, запросу необходим заголовок `Content-Length`, сообщающий серверу ожидаемый размер содержимого в теле запроса.

Из-за обработки аргументов и дополнительных заголовков невозможно посредством функции `fopen()` построить запрос POST. Если недоступны ни расширение cUrl, ни класс `HTTP_Request`, то обратитесь к функции `pc_post_request()`, показанной в примере 1.1, которая осуществляет соединение с удаленным веб-сервером с помощью функции `fsockopen()`.

Пример 11.1. pc_post_request( )

```
function pc_post_request($host,$url,$content='') {
    $timeout = 2;
    $a = array();
    if (is_array($content)) {
        foreach ($content as $k => $v) {
            array_push($a,urlencode($k).'='.urlencode($v));
        }
    }
    $content_string = join('&',$a);
    $content_length = strlen($content_string);
    $request_body = "POST $url HTTP/1.0
Host: $host
Content-type: application/x-www-form-urlencoded
Content-length: $content_length

$content_string";

    $sh = fsockopen($host,80,&$errno,&$errstr,$timeout)
        or die("can't open socket to $host: $errno $errstr");

    fputs($sh,$request_body);
    $response = '';
    while (! feof($sh)) {
        $response .= fread($sh,16384);
    }
    fclose($sh) or die("Can't close socket handle: $php_errormsg");
```

```
    list($response_headers,$response_body) = explode("\r\n\r\n",$response,2);
    $response_header_lines = explode("\r\n",$response_headers);

    // первая строка заголовков представляет код ответа HTTP
    $http_response_line = array_shift($response_header_lines);
    if (preg_match('@^HTTP/[0-9]\.[0-9] ([0-9]{3})@',$http_response_line,
                   $matches)) {
        $response_code = $matches[1];
    }

    // помещаем оставшиеся части заголовков в массив
    $response_header_array = array();
    foreach ($response_header_lines as $header_line) {
        list($header,$value) = explode(': ',$header_line,2);
        $response_header_array[$header] = $value;
    }

    return array($response_code,$response_header_array,$response_body);
}
```

Функцию `pc_post_request()` надо вызывать так:

```
list($code,$headers,$body) = pc_post_request('www.example.com','/submit.php',
                                             array('monkey' => 'uncle',
                                                   'rhino' => 'aunt'));
```

Получение доступа к URL с помощью метода POST вместо метода GET особенно полезно, если URL очень длинный, более 200 символов или около того. Спецификация HTTP 1.1 в RFC 2616 не ограничивает максимальную длину URL, поэтому поведение различных веб- и прокси-серверов отличается. Если вы извлекаете содержимое URL с помощью метода GET и получаете неожиданные результаты или результаты с кодом статуса 414 («Request-URI Too Long» (Запрос-URI слишком длинный)), то измените метод запроса на POST.

См. также

Рецепт 11.1 о получении содержимого URL с помощью метода GET; документацию по функции `curl_setopt()` на *http://www.php.net/curl-setopt* и по функции `fsockopen()` на *http://www.php.net/fsockopen*; о классе PEAR `HTTP_Request` на *http://pear.php.net/package-info.php?package=HTTP_Request*; RFC 2616 на *http://www.faqs.org/rfcs/rfc2616.html*.

11.3. Получение содержимого URL, если требуется отправить cookies

Задача

Необходимо получить страницу, которая требует посылки cookie вместе с запросом к ней.

Решение

Используйте расширение cURL и параметр CURLOPT_COOKIE:

```
$c = curl_init('http://www.example.com/needs-cookies.php');
curl_setopt($c, CURLOPT_VERBOSE, 1);
curl_setopt($c, CURLOPT_COOKIE, 'user=ellen; activity=swimming');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
$page = curl_exec($c);
curl_close($c);
```

Если расширение cURL недоступно, то используйте метод addHeader() класса HTTP_Request из PEAR:

```
require 'HTTP/Request.php';

$r = new HTTP_Request('http://www.example.com/needs-cookies.php');
$r->addHeader('Cookie','user=ellen; activity=swimming');
$r->sendRequest();
$page = $r->getResponseBody();
```

Обсуждение

Cookies посылаются на сервер в заголовке Cookie запроса. В расширении cURL есть специальный параметр для cookie, но применяя класс HTTP_Request, необходимо добавлять заголовок Cookie точно так же, как и другие заголовки запроса. Несколько значений cookie посылаются списком с точкой с запятой в конце. Примеры в разделе «Решение» посылают два cookies: один с именем user и значением ellen, а другой с именем activity и значением swimming.

Чтобы запросить страницу, которая устанавливает cookies, а затем посылает последующие запросы, содержащие эти только что установленные cookies, используйте возможность расширения cURL, называемую «cookie jar» (банка для cookie). В первом запросе присваиваем параметру CURLOPT_COOKIEJAR имя файла, хранящего cookies. В последующих запросах присваиваем параметру CURLOPT_COOKIEFILE то же самое имя файла, а расширение cURL читает cookies из файла и посылает их вместе с запросом. Это особенно полезно, когда есть последовательность запросов, первый из которых регистрируется на сайте, устанавливающем cookies сессии или cookies аутентификации и требующем, чтобы остальные запросы содержали эти установленные cookies:

```
$cookie_jar = tempnam('/tmp','cookie');

// регистрируемся
$c = curl_init('https://bank.example.com/
login.php?user=donald&password=b1gmoney$');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_COOKIEJAR, $cookie_jar);
$page = curl_exec($c);
curl_close($c);
```

```
// извлекаем баланс счета
$c = curl_init('http://bank.example.com/balance.php?account=checking');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_COOKIEFILE, $cookie_jar);
$page = curl_exec($c);
curl_close($c);

// делаем депозит
$c = curl_init('http://bank.example.com/deposit.php');
curl_setopt($c, CURLOPT_POST, 1);
curl_setopt($c, CURLOPT_POSTFIELDS, 'account=checking&amount=122.44');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_COOKIEFILE, $cookie_jar);
$page = curl_exec($c);
curl_close($c);

// удаляем cookie jar
unlink($cookie_jar) or die("Can't unlink $cookie_jar");
```

Определяя место хранения cookie jar, соблюдайте осторожность. Это должна быть область, куда веб-сервер имеет право записывать, но если другие пользователи имеют возможность читать этот файл, то они смогут незаконно получить идентификационные параметры, хранящиеся в cookies.

См. также

Документацию по функции `curl_setopt()` на *http://www.php.net/curl-setopt*; о классе PEAR `HTTP_Request` на *http://pear.php.net/package-info.php?package=HTTP_Request*

11.4. Получение содержимого URL, требующее отправки заголовков

Задача

Необходимо получить содержимое URL, требующего посылки специальных заголовков вместе с запросом к данной странице.

Решение

Для этого применяется расширение cURL и параметр `CURLOPT_HTTPHEADER`:

```
$c = curl_init('http://www.example.com/special-header.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_HTTPHEADER, array('X-Factor: 12', 'My-Header:
Bob'));
$page = curl_exec($c);
curl_close($c);
```

Если расширение cURL недоступно, то применяйте метод `addHeader()` класса `HTTP_Request`:

```
require 'HTTP/Request.php';

$r = new HTTP_Request('http://www.example.com/special-header.php');
$r->addHeader('X-Factor',12);
$r->addHeader('My-Header','Bob');
$r->sendRequest();
$page = $r->getResponseBody();
```

Обсуждение

Расширение cURL имеет специальные параметры, `CURLOPT_REFERER` и `CURLOPT_USERAGENT`, позволяющие устанавливать заголовки запроса `Referer` и `User-Agent`:

```
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_VERBOSE, 1);
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_REFERER, 'http://www.example.com/form.php');
curl_setopt($c, CURLOPT_USERAGENT, 'CURL via PHP');
$page = curl_exec($c);
curl_close($c);
```

См. также

Рецепт 11.13, объясняющий, почему слово «referrer» часто ошибочно записывается как «referer» в контексте веб-программирования; документацию по функции `curl_setopt()` на *http://www.php.net/curl-setopt*; о классе PEAR `HTTP_Request` на *http://pear.php.net/package-info.php?package=HTTP_Request*.

11.5. Получение содержимого HTTPS URL

Задача

Необходимо получить доступ к содержимому защищенного URL.

Решение

С этой целью применяется расширение cURL для HTTPS URL:

```
$c = curl_init('https://secure.example.com/accountbalance.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
$page = curl_exec($c);
curl_close($c);
```

Обсуждение

Для того чтобы получить содержимое защищенного URL, расширению cURL необходим доступ к какой-нибудь библиотеке SSL, такой

как OpenSSL. Эта библиотека должна быть доступна во время построения PHP и расширения cURL. Не считая этих дополнительных требований к библиотеке, расширение cURL обращается с защищенными URL точно так же, как и с обычными ссылками. Для защищенных запросов расширение cURL предоставляет те же самые возможности, например изменение метода запроса или добавление POST-данных.

См. также

О проекте OpenSSL на *http://www.openssl.org/*.

11.6. Отладка обмена заголовками HTTP

Задача

Необходимо проанализировать HTTP-запрос броузера к серверу и соответствующий HTTP-ответ. Например, сервер не выдает ожидаемого ответа на определенный запрос, поэтому требуется точно определить, какие компоненты запрашивались.

Решение

Если запросы простые, надо соединиться с веб-сервером с помощью программы *telnet* и ввести с клавиатуры следующие заголовки запроса:

```
% telnet www.example.com 80
Trying 10.1.1.1...
Connected to www.example.com.
Escape character is '^]'.
GET / HTTP/1.0
Host: www.example.com

HTTP/1.1 200 OK
Date: Sat, 17 Aug 2002 06:10:19 GMT
Server: Apache/1.3.26 (UNIX) PHP/4.2.2 mod_ssl/2.8.9 OpenSSL/0.9.6d
X-Powered-By: PHP/4.2.2
Connection: close
Content-Type: text/html

// ... основная часть страницы ...
```

Обсуждение

Когда вы печатаете от руки заголовки запроса, веб-сервер не знает, что это именно вы печатаете, а не броузер представляет запрос. Однако некоторые веб-серверы определенное время ожидают запроса, поэтому иногда удобнее заблаговременно набрать запрос, а затем вставить его в окно программы *telnet*. Первая строка запроса содержит метод запроса (`GET`), пробел и путь к требуемому файлу (`/`), а затем пробел и используемый протокол (`HTTP/1.0`). Следующая строка, заголовок `Host`, сообща-

ет серверу, какой виртуальный сервер использовать, если несколько серверов используют один и тот же IP-адрес. Пустая строка говорит серверу, что запрос завершен; тогда он выдает свой ответ: сначала заголовки, потом пустую строку, а затем основное содержание ответа.

Вставка текста в окно программы *telnet* может быть утомительным занятием, а еще труднее таким образом делать запросы методом POST. При посылке запроса с использованием класса `HTTP_Request` можно получить заголовки и тело ответа с помощью методов `getResponseHeader()` и `getResponseBody()`:

```
require 'HTTP/Request.php';

$r = new HTTP_Request('http://www.example.com/submit.php');
$r->setMethod(HTTP_REQUEST_METHOD_POST);
$r->addPostData('monkey','uncle');
$r->sendRequest();

$response_headers = $r->getResponseHeader();
$response_body    = $r->getResponseBody();
```

Чтобы получить определенный заголовок ответа, передайте имя заголовка функции `getResponseHeader()`. Без аргумента метод `getResponseHeader()` возвращает массив, содержащий все заголовки ответа. Класс `HTTP_Request` не сохраняет выходящие запросы в переменной, но его можно реконструировать с помощью вызова частного метода `_buildRequest()`:

```
require 'HTTP/Request.php';

$r = new HTTP_Request('http://www.example.com/submit.php');
$r->setMethod(HTTP_REQUEST_METHOD_POST);
$r->addPostData('monkey','uncle');

print $r->_buildRequest();
```

Напечатанный запрос выглядит так:

```
POST /submit.php HTTP/1.1
User-Agent: PEAR HTTP_Request class ( http://pear.php.net/ )
Content-Type: application/x-www-form-urlencoded
Connection: close
Host: www.example.com
Content-Length: 12

monkey=uncle
```

При использовании расширения cURL для включения заголовков ответа в вывод функции `curl_exec()` установите параметр `CURLOPT_HEADER`:

```
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_HEADER, 1);
curl_setopt($c, CURLOPT_POST, 1);
curl_setopt($c, CURLOPT_POSTFIELDS, 'monkey=uncle&rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
$response_headers_and_page = curl_exec($c);
curl_close($c);
```

Чтобы записать заголовки ответа в файл на диске, откройте дескриптор файла с помощью функции fopen() и установите для этого дескриптора файла параметр CURLOPT_WRITEHEADER:

```
$fh = fopen('/tmp/curl-response-headers.txt','w') or die($php_errormsg);
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_POST, 1);
curl_setopt($c, CURLOPT_POSTFIELDS, 'monkey=uncle&rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_WRITEHEADER, $fh);
$page = curl_exec($c);
curl_close($c);
fclose($fh) or die($php_errormsg);
```

Параметр CURLOPT_VERBOSE модуля cURL заставляет функции curl_exec() и curl_close() выводить отладочную информацию в поток стандартных ошибок, включая содержимое запроса:

```
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_VERBOSE, 1);
curl_setopt($c, CURLOPT_POST, 1);
curl_setopt($c, CURLOPT_POSTFIELDS, 'monkey=uncle&rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
$page = curl_exec($c);
curl_close($c);
```

В результате будет напечатано:

```
* Connected to www.example.com (10.1.1.1)
> POST /submit.php HTTP/1.1
Host: www.example.com
Pragma: no-cache
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Content-Length: 23
Content-Type: application/x-www-form-urlencoded

monkey=uncle&rhino=aunt* Connection #0 left intact
* Closing connection #0
```

Поскольку расширение cURL выводит отладочную информацию в поток стандартных ошибок, а не в стандартный поток вывода, он не может быть захвачен буфером вывода, подобно тому как в рецепте 10.10 это делается с помощью функции print_r(). Однако чтобы направить отладочную информацию в файл, можно открыть дескриптор файла на запись и установить для этого дескриптора параметр CURLOUT_STDERR:

```
$fh = fopen('/tmp/curl.out','w') or die($php_errormsg);
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_VERBOSE, 1);
curl_setopt($c, CURLOPT_POST, 1);
curl_setopt($c, CURLOPT_POSTFIELDS, 'monkey=uncle&rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_STDERR, $fh);
```

```
$page = curl_exec($c);
curl_close($c);
fclose($fh) or die($php_errormsg);
```

См. также

Рецепт 10.10 о буферизации вывода; документацию по функции `curl_setopt()` на *http://www.php.net/curl-setopt*; о классе PEAR `HTTP_Request` на *http://pear.php.net/package-info.php?package=HTTP_Request*; RFC 2616 на *http://www.faqs.org/rfcs/rfc2616.html*, в котором определен синтаксис HTTP-запроса.

11.7. Выделение информации на веб-странице

Задача

Требуется показать страницу, например результаты поиска, подсветив при этом некоторые слова.

Решение

Вызовите функцию `preg_replace()` с массивом шаблонов и массивом заместителей:

```
$patterns = array('\bdog\b/', '\bcat\b');
$replacements = array('<b style="color:black;background-color=#FFFF00">
                        dog</b>',
                      '<b style='color:black;background-color=#FF9900">
                        cat</b>');
while ($page) {
    if (preg_match('{^([^<]*)?(</?[^>]+?>)?(.*)$}',$page,$matches)) {
        print preg_replace($patterns,$replacements,$matches[1]);
        print $matches[2];
        $page = $matches[3];
    }
}
```

Обсуждение

Регулярное выражение, используемое функцией `preg_match()`, сопоставляет весь возможный текст до тега HTML, затем тег, а затем остальное содержимое. Текст до тега HTML подсвечивается, тег выводится без какого-либо выделения, а остальное содержимое сопоставляется таким же образом. Это предотвращает выделение слов внутри тегов HTML (например, в URL или в тексте атрибута `alt`), что могло бы привести к некорректному отображению страницы.

Следующая программа извлекает содержимое URL и помещает его в переменную `$url` и подсвечивает слова из массива `$words`. Слова не выделяются, если они являются частью более длинного слова, поскольку

они соответствуют Perl-совместимому оператору регулярного выражения \b для определения границ слова.

```
$colors = array('FFFF00','FF9900','FF0000','FF00FF',
                '99FF33','33FFCC','FF99FF','00CC33');

// строим поиск и заменяем шаблоны регулярным выражением
$patterns = array();
$replacements = array();
for ($i = 0, $j = count($words); $i < $j; $i++) {
    $patterns[$i] = '/\b'.preg_quote($words[$i], '/').'\b/';
    $replacements[$i] = '<b style="color:black;background-color:#' .
                        $colors[$i % 8] .'">' . $words[$i] . '</b>';
}

// получаем страницу
$fh = fopen($url,'r') or die($php_errormsg);
while (! feof($fh)) {
    $s .= fread($fh,4096);
}
fclose($fh);

if ($j) {
    while ($s) {
        if (preg_match('{^([^<]*)?(</?[^>]+?>)?(.*)$}s',$s,$matches)) {
            print preg_replace($patterns,$replacements,$matches[1]);
            print $matches[2];
            $s = $matches[3];
        }
    }
} else {
    print $s;
}
```

См. также

Рецепт 13.7 об извлечении текста, заключенного в теги HTML; документацию по функции preg_match() на *http://www.php.net/preg-match* и по функции preg_replace() на *http://www.php.net/preg-replace*.

11.8. Извлечение ссылок из HTML-файла

Задача

Необходимо извлечь все URL, определенные в документе HTML.

Решение

Это можно сделать при помощи функции pc_link_extractor(), показанной в примере 11.2.

Пример 11.2. pc_link_extractor( )

```
function pc_link_extractor($s) {
  $a = array();
  if (preg_match_all(
                 '/<a\s+.*?href=[\"\']?([^\"\' >]*)[\"\']?[^>]*>(.*?)<\a>/i',
                 $s,$matches,PREG_SET_ORDER)) {
    foreach($matches as $match) {
      array_push($a,array($match[1],$match[2]));
    }
  }
  return $a;
}
```

Например:

```
$links = pc_link_extractor($page);
```

Обсуждение

Функция `pc_link_extractor()` возвращает массив. Каждый элемент этого массива сам является двухэлементным массивом. Первый аргумент – это адрес гиперссылки, а второй аргумент – ее текст. Например:

```
$links=<<<END
Click <a href="http://www.oreilly.com">here</a> to visit a computer book
publisher. Click <a href="http://www.sklar.com">over here</a> to visit
a computer book author.
END;

$a = pc_link_extractor($links);
print_r($a);
Array
(
    [0] => Array
        (
            [0] => http://www.oreilly.com
            [1] => here
        )
    [1] => Array
        (
            [0] => http://www.sklar.com
            [1] => over here
        )
)
```

Регулярное выражение в функции `pc_link_extractor()` не будет работать для всех ссылок, например для тех, которые построены с помощью JavaScript, или адресов, представленных в шестнадцатеричных кодах, но оно должно функционировать для большинства корректно отформатированных документов HTML.

См. также

Рецепт 13.7, где приведена информация о выделении текста в тегах HTML; документацию по функции `preg_match_all()` на *http://www.php.net/preg-match-all*.

11.9. Преобразование ASCII в HTML

Задача

Необходимо преобразовать простой текст в корректно отформатированный документ HTML.

Решение

Во-первых, закодируйте элементы с помощью функции `htmlentities()`,[1] затем преобразуйте текст в различные HTML-структуры. Функция `pc_ascii2html()`, показанная в примере 11.3, выполняет основные преобразования для ссылок и символов конца абзаца.

Пример 11.3. pc_ascii2html( )

```
function pc_ascii2html($s) {
  $s = htmlentities($s);
  $grafs = split("\n\n",$s);
  for ($i = 0, $j = count($grafs); $i < $j; $i++) {
    // Ссылка на то, что выглядит как http или ftp URL
    $grafs[$i] = preg_replace('/((ht|f)tp:\/\/[^\s&]+)/',
                              '<a href="$1">$1</a>',$grafs[$i]);

    // Ссылка на почтовые адреса
    $grafs[$i] = preg_replace('/[^@\s]+@([-a-z0-9]+\.)+[a-z]{2,}/i',
        '<a href="mailto:$1">$1</a>',$grafs[$i]);

    // Начало нового абзаца
    $grafs[$i] = '<p>'.$grafs[$i].'</p>';
  }
  return join("\n\n",$grafs);
}
```

Обсуждение

Чем больше вы знаете о строении ASCII-текста, тем лучше может быть HTML-преобразование. Например, если выделение обозначается звездочками (*asterisks*) или символами косой черты (/slashes/) вокруг слова, то можно добавить правила, реализующие это следующим образом:

```
$grafs[$i] = preg_replace('/(\A|\s)\*([^*]+)\*(\s|\z)/',
                          '$1<b>$2</b>$3',$grafs[$i]);
```

[1] Для кириллического текста `htmlspecialchars()`. – *Примеч. науч. ред.*

```
$grafs[$i] = preg_replace('{(\A|\s)/([^/]+)/(\s|\z)}',
                          '$1<i>$2</i>$3',$grafs[$i]);
```

См. также

Документацию по функции `preg_replace()` на *http://www.php.net/preg-replace*.

11.10. Преобразование HTML в ASCII

Задача

Необходимо преобразовать документ HTML в читаемый, форматированный ASCII-текст.

Решение

Если доступны внешние программы, преобразующие HTML в ASCII, такие как *lynx*, то вызовите их, например так:

```
$file = escapeshellarg($file);
$ascii = `lynx -dump $file`;
```

Обсуждение

Если внешняя программа форматирования недоступна, то функция `pc_html2ascii()`, показанная в примере 11.4, обрабатывает приемлемое подмножество HTML-элементов (без таблиц, фреймов и т. д.).

Пример 11.4. pc_html2ascii( )

```
function pc_html2ascii($s) {
  // конвертируем ссылки
  $s = preg_replace('/<a\s+.*?href="?([^\" >]*)"?[^>]*>(.*?)<\/a>/i',
                    '$2 ($1)', $s);

  // конвертируем <br>, <hr>, <p>, <div> в символы конца строки
  $s = preg_replace('@<(b|h)r[^>]*>@i',"\n",$s);
  $s = preg_replace('@<p[^>]*>@i',"\n\n",$s);
  $s = preg_replace('@<div[^>]*>(.*)</div>@i","\n".'$1'."\n",$s);

  // конвертируем полужирный шрифт и курсив
  $s = preg_replace('@<b[^>]*>(.*?)</b>@i','*$1*',$s);
  $s = preg_replace('@<i[^>]*>(.*?)</i>@i','/$1/',$s);

  // декодируем поименованные элементы
  $s = strtr($s,array_flip(get_html_translation_table(HTML_ENTITIES)));

  // декодируем нумерованные элементы
  $s = preg_replace('//e','chr(\\1)',$s);

  // удаляем все оставшиеся теги
  $s = strip_tags($s);
```

```
    return $s;
}
```

См. также

Дополнительную информацию по функции `get_html_translation_table()` в рецепте 9.8; документацию по функции `preg_replace()` на *http://www.php.net/preg-replace*, по функции `get_html_translation_table()` на *http://www.php.net/get-html-translation-table* и по функции `strip_tags()` на *http://www.php.net/strip-tags*.

11.11. Удаление тегов HTML и PHP

Задача

Необходимо удалить теги HTML и PHP из строки или файла.

Решение

Удаление тегов HTML и PHP из строки выполняется посредством функции `strip_tags()`:

```
$html = '<a href="http://www.oreilly.com">I <b>love computer books.</b></a>';
print strip_tags($html);
I love computer books.
```

Функция `fgetss()` позволяет удалять их из файла по мере чтения строк:

```
$fh = fopen('test.html','r') or die($php_errormsg);
while ($s = fgetss($fh,1024)) {
    print $s;
}
fclose($fh)                  or die($php_errormsg);
```

Обсуждение

Функция `fgetss()` удобна, когда требуется удалить теги из файла в процессе его чтения, но ее вызов может привести к сбою, если теги охватывают несколько строк или весь буфер, в который функция `fgetss()` читает из файла. За счет увеличения объема используемой памяти чтение всего файла в строку обеспечивает лучшие результаты:

```
$no_tags = strip_tags(join('',file('test.html')));
```

И функции `strip_tags()`, и функции `fgetss()` можно приказать не удалять определенные теги, указывая их в последнем аргументе.[1] Опреде-

[1] Однако надо помнить, что злонамеренный пользователь может вставить вредоносный сценарий и в атрибуты совершенно невинных тегов. – *Примеч. науч. ред.*

ление тега чувствительно к регистру, а для пары тегов необходимо указывать только открывающий тег. Например, следующий фрагмент удаляет из переменной `$html` все теги, за исключением `<b></b>`:

```
$html = '<a href="http://www.oreilly.com">I <b>love</b> computer books.</a>';
print strip_tags($html,'<b>');
I <b>love</b> computer books.
```

См. также

Документацию по функции `strip_tags()` на *http://www.php.net/strip-tags* и по функции `fgetss()` на *http://www.php.net/fgetss*.

11.12. Использование шаблонов системы Smarty

Задача

Необходимо разделить код и дизайн страниц. Дизайнеры смогут работать над файлами HTML, не касаясь программы PHP, а программисты смогут работать над PHP-кодом, не заботясь о дизайне.

Решение

Применяйте систему шаблонов. Одна из систем, с которой нетрудно работать, называется Smarty. В шаблоне Smarty строки в фигурных скобках заменяются новыми значениями:

```
Hello, {$name}
```

PHP-программа, создающая страницу, устанавливает переменные, а затем выводит шаблон, как показано ниже:

```
require 'Smarty.class.php';

$smarty = new Smarty;
$smarty->assign('name','Ruby');
$smarty->display('hello.tpl');
```

Обсуждение

Следующий шаблон предназначен для отображения строк, извлеченных из базы данных:

```
<html>
<head><title>cheeses</title></head>
<body>
<table border="1">
<tr>
  <th>cheese</th>
  <th>country</th>
  <th>price</th>
</tr>
```

```
{section name=id loop=$results}
<tr>
  <td>{$results[id]->cheese}</td>
  <td>{$results[id]->country}</td>
  <td>{$results[id]->price}</td>
</tr>
{/section}
</table>
</body>
</html>
```

Далее показан соответствующий PHP-файл, который загружает информацию из базы данных, а затем показывает шаблон, хранящийся в файле *food.tpl*:

```
require 'Smarty.class.php';

mysql_connect('localhost','test','test');
mysql_select_db('test');

$r = mysql_query('SELECT * FROM cheese');
while ($ob = mysql_fetch_object($r)) {
    $ob->price = sprintf('$%.02f',$ob->price);
    $results[] = $ob;

}
$smarty = new Smarty;
$smarty->assign('results',$results);
$smarty->display('food.tpl');
```

После включения базового класса движка шаблонов (*Smarty.class.php*), результаты извлекаются из базы данных, форматируются, а затем помещаются в массив. Для того чтобы сгенерировать шаблонную страницу, достаточно создать новый объект `$smarty`, сказать ему, чтобы он обратил внимание на переменную `$results`, а затем приказать объекту `$smarty` показать шаблон.

Систему Smarty легко инсталлировать – достаточно скопировать новые файлы в ваш каталог `include_path` и создать новые каталоги. Полную инструкцию можно найти на *http://smarty.php.net/manual/en/installing.smarty.basic.html*. Работая с системой Smarty, соблюдайте дисциплину, это поможет сохранить основное назначение шаблонов – разделять логику и представление. Движок шаблонов имеет свой собственный язык сценариев, который можно использовать для получения значения переменных, выполнения циклов и реализации других простых алгоритмов. Постарайтесь свести его использование в своих шаблонах к минимуму, а всю логику программы поместить в PHP-файл.

См. также

Домашнюю страницу системы Smarty на *http://smarty.php.net/*.

11.13. Анализ файла протокола веб-сервера

Задача

Необходимо сделать расчеты на основании информации, содержащейся в файле протокола доступа к веб-серверу.

Решение

Откройте файл и проанализируйте каждую строку с помощью регулярного выражения, соответствующего формату файла протокола. Приведенное ниже регулярное выражение соответствует комбинированному формату протокола NCSA:

```
$pattern = '/^([^ ]+) ([^ ]+) ([^ ]+) (\[[^\]]+\]) "(.*) (.*) (.*)" ([0-9\-]+)
    ([0-9\-]+) "(.*)" "(.*)"$/';
```

Обсуждение

Эта программа анализирует строки в комбинированном формате протокола (Combined Log Format) NCSA и показывает список страниц, отсортированных по количеству запросов к каждой странице:

```
$log_file = '/usr/local/apache/logs/access.log';
$pattern = '/^([^ ]+) ([^ ]+) ([^ ]+) (\[[^\]]+\]) "(.*) (.*) (.*)" ([0-9\-]+)
    ([0-9\-]+) "(.*)" "(.*)"$/';

$fh = fopen($log_file,'r') or die($php_errormsg);
$i = 1;
$requests = array();
while (! feof($fh)) {
    // читаем каждую строку и удаляем начальные и конечные пробельные символы
    if ($s = trim(fgets($fh,16384))) {
        // сравниваем строку с шаблоном
        if (preg_match($pattern,$s,$matches)) {
            /* помещаем каждую совпавшую часть в переменную
             * с соответствующим именем */
            list($whole_match,$remote_host,$logname,$user,$time,
                 $method,$request,$protocol,$status,$bytes,$referer,
                 $user_agent) = $matches;
             // ведем учет каждого запроса
            $requests[$request]++;
        } else {
            // выводим предупреждение, если строка не соответствует шаблону
            error_log("Can't parse line $i: $s");
        }
    }
    $i++;
}
fclose($fh) or die($php_errormsg);

// сортируем массив (в обратном порядке)  по количеству запросов
arsort($requests);
```

```
// выводим отформатированные  результаты
foreach ($requests as $request => $accesses) {
    printf("%6d   %s\n",$accesses,$request);
}
```

Шаблон, используемый в функции `preg_match()`, соответствует строкам в комбинированном формате протокола, таким как:

```
10.1.1.162 - david [20/Jul/2001:13:05:02 -0400] "GET /sklar.css HTTP/1.0" 200
278 "-" "Mozilla/4.77 [en] (WinNT; U)"
10.1.1.248 - - [14/Mar/2002:13:31:37 -0500] "GET /php-cookbook/colors.html
HTTP/1.1" 200 460 "-" "Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)"
```

В первой строке значение `10.1.1.162` представляет IP-адрес, с которого пришел запрос. В зависимости от настроек сервера, вместо него может стоять имя хоста. Когда значения массива `$matches` присваиваются списку отдельных переменных, то имя хоста запоминается в переменной `$remote_host`. Следующий дефис (-) означает, что удаленный хост не предоставил имя пользователя через *identd*,[1] поэтому переменной `$logname` присвоено значение «-».

Строка `david` содержит имя пользователя, предоставленное броузером с помощью базовой аутентификации HTTP, и присваивается переменной `$user`. Дата и время запроса, сохраненные в переменной `$time`, заключены в скобки. Этот формат даты и времени не распознается функцией `strtotime()`, поэтому, если вы хотите сделать вычисления, основанные на дате и времени запроса, то должны выполнить обработку для выделения каждой части форматированной строки времени. Следом, в кавычках, стоит первая строка запроса. Она состоит из метода (GET, POST, HEAD и т. д.), хранимого в переменной `$method`; запрошенного URI, хранящегося в переменной `$request`, и протокола, находящегося в переменной `$protocol`. Для запросов GET строка запроса является частью URI. Для запросов POST тело запроса, содержащее переменные, в протокол не заносится.

После запроса идет статус запроса, хранящийся в переменной `$status`. Статус `200` означает, что запрос успешно выполнен. После статуса находится размер ответа в байтах, присвоенный переменной `$bytes`. Последние два элемента строки, причем каждый из них в кавычках, это страница, с которой сделана ссылка, если таковая имеется, хранящаяся в переменной `$referer`,[2] и строка пользовательского агента, иденти-

---

[1] *identd*, определенная в RFC 1413, считается хорошим способом удаленной идентификации пользователей. Однако он не отличается защищенностью и надежностью. Квалифицированное объяснение можно найти на *http://www.clock.org/~fair/opinion/identd.html*.

[2] Правильным написанием этого слова является «referrer». Однако в оригинальной спецификации HTTP (RFC 1945) это слово ошибочно записано как «referer», поэтому часто в контексте встречается написание с тремя «r».

фицирующая броузер, который послал запрос, и хранящаяся в переменной `$user_agent`.

Как только строка файла протокола разобрана на отдельные переменные, можно делать необходимые вычисления. В этом случае достаточно сохранить счетчик, определяющий количество запросов каждого URI, в массиве `$requests`. После выполнения цикла по всем строкам файла выведите отсортированный и отформатированный список запросов и значений счетчиков.

Такой способ определения статистики протоколов доступа к веб-серверу легок, но не очень гибок. Требуется модификация программы для различных видов отчетов, ограниченных диапазонов данных, форматирования отчетов и многих других целей. Наилучшим решением для получения всесторонней статистики веб-сайта является применение таких программ, как *analog*, свободно доступной на *http://www.analog.cx*. Она предоставляет много типов отчетов и настроек конфигурации, которые должны удовлетворить практически любое ваше пожелание.

См. также

Документацию по функции `preg_match()` на *http://www.php.net/preg-match*; информацию об общепринятых форматах файла протокола на *http://httpd.apache.org/docs/logs.html*.

11.14. Программа: обнаружение устаревших сылок

Программа *stale-links.php* выдает список ссылок на странице и их статус. Она сообщает, действительны ли ссылки, не перемещены ли они куда-нибудь, или они уже недействительны. Запустите программу, передав ей URL для сканирования ссылок:

```
% stale-links.php http://www.oreilly.com/
http://www.oreilly.com/index.html: OK
http://www.oreillynet.com: OK
http://conferences.oreilly.com: OK
http://international.oreilly.com: OK
http://safari.oreilly.com: MOVED: mainhom.asp?home
...
```

Для получения страниц программа *stale-links.php* использует расширение cURL. Сначала она извлекает содержимое URL, указанного в командной строке. После получения страницы программа извлекает ссылки, имеющиеся на этой странице, с помощью функции `pc_link_extractor()` из рецепта 11.8. Затем, после присоединения базового URL в начало каждой ссылки, если это необходимо, запрашивается содержимое ссылки. Нам нужны лишь заголовки ответа, поэтому мы выбираем метод HEAD вместо метода GET, устанавливая параметр

CURLOPT_NOBODY. Если установить параметр CURLOPT_HEADER, то функция curl_exec() включит заголовки ответа в возвращаемую ею строку. На основании кода ответа выводится статус ссылки вместе с новым местоположением, если оно было изменено.

Пример 11.5. stale-links.php

```
function_exists('curl_exec') or die('CURL extension required');

function pc_link_extractor($s) {
    $a = array();
    if (preg_match_all(
                '/<A\s+.*?HREF=[\"\']?([^\"\' >]*)[\"\']?[^>]*>(.*?)<\/A>/i',
                $s,$matches,PREG_SET_ORDER)) {
        foreach($matches as $match) {
            array_push($a,array($match[1],$match[2]));
        }
    }
    return $a;
}

$url = $_SERVER['argv'][1];

// извлекаем содержимое URL
$c = curl_init($url);
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($c, CURLOPT_FOLLOWLOCATION,1);
$page = curl_exec($c);
$info = curl_getinfo($c);
curl_close($c);

// определяем базовый url из url
// при этом не обращаем внимания на тег <base> на этой странице
$url_parts = parse_url($info['url']);
if ('' == $url_parts['path']) { $url_parts['path'] = '/'; }
$base_path = preg_replace('<^(.*/)([^/]*)$>','\\1',$url_parts['path']);
$base_url = sprintf('%s://%s%s%s',
                    $url_parts['scheme'],
                    ($url_parts['username'] || $url_parts['password']) ?
                    "$url_parts[username]:$url_parts[password]@" : '',
                    $url_parts['host'],
                    $url_parts['path']);

// запоминаем ссылки, которые мы посетили, чтобы не посещать
// их более одного раза
$seen_links = array();

if ($page) {
    $links = pc_link_extractor($page);
    foreach ($links as $link) {
        // вычисляем относительные ссылки
        if (! (preg_match('{^(http|https|mailto):}',$link[0]))) {
            $link[0] = $base_url.$link[0];
        }
```

```
        // пропускаем данную ссылку, если мы ее уже видели
        if ($seen_links[$link[0]]) {
            continue;
        }

        // отмечаем данную ссылку как просмотренную
        $seen_links[$link[0]] = true;

        // выводим содержимое ссылки, на которой находимся
        print $link[0].': ';
        flush();

        // посещаем ссылку
        $c = curl_init($link[0]);
        curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($c, CURLOPT_NOBODY, 1);
        curl_setopt($c, CURLOPT_HEADER, 1);
        $link_headers = curl_exec($c);
        $curl_info = curl_getinfo($c);
        curl_close($c);

        switch (intval($curl_info['http_code']/100)) {
        case 2:
            // коды ответа 2xx означают, что со страницей все в порядке
            $status = 'OK';
            break;
        case 3:
            // коды ответа 3xx означают переадресацию
            $status = 'MOVED';
            if (preg_match('/^Location: (.*)$/m',$link_headers,$matches)) {
                $location = trim($matches[1]);
                $status .= ": $location";
            }
            break;
        default:
            // остальные коды ответа означают ошибки
            $status = "ERROR: $curl_info[http_code]";
            break;
        }

        print "$status\n";
    }
}
```

11.15. Программа: Обнаружение свежих ссылок

Пример 11.6, *fresh-links.php*, – это модификация программы из рецепта 11.14, которая выдает список ссылок и время их последнего изменения. Если сервер, на котором находится ссылка, не сообщает время последнего изменения, то в качестве времени последнего изменения программа возвращает время запроса URL. Если запрос программой URL заканчивается неудачей, то она выводит код статуса, который про-

грамма получила при попытке получения содержимого URL. Запустите программу, передав ей URL для просмотра ссылок:

```
% fresh-links.php http://www.oreilly.com
http://www.oreilly.com/index.html: Fri Aug 16 16:48:34 2002
http://www.oreillynet.com: Mon Aug 19 10:18:54 2002
http://conferences.oreilly.com: Fri Aug 16 19:41:46 2002
http://international.oreilly.com: Fri Mar 29 18:06:32 2002
http://safari.oreilly.com: 302
http://www.oreilly.com/catalog/search.html: Tue Apr  2 19:05:57 2002
http://www.oreilly.com/oreilly/press/: 302
...
```

Этот вывод является результатом запуска программы примерно в 10:20 A.M. EDT 19 августа 2002 года. Ссылка на *http://www.oreillynet.com* очень свежая, а другие имеют различный возраст. За ссылкой на *http://www.oreilly.com/oreilly/press/* нет времени последней модификации; вместо него стоит код статуса (302). Это означает, что ссылка была перемещена, как говорит нам результат работы *stale-links.php* в рецепте 11.14.

Программа обнаружения свежих ссылок концептуально почти идентична программе поиска устаревших ссылок. Она использует ту же самую функцию `pc_link_extractor()` из рецепта 11.9; однако получение содержимого URL в программе реализовано на основе класса `HTTP_Request` вместо расширения cURL. Код для извлечения базового URL, указанного в командной строке, находится внутри цикла, так что он может следовать любой возвращенной переадресации.

Получив страницу, программа вызывает функцию `pc_link_extractor()`, чтобы сгенерировать список ссылок, имеющихся на странице. Затем, после присоединения базового URL в начало каждой ссылки, если это необходимо, для каждой ссылки, найденной на исходной странице, вызывается функция `sendRequest()`. Нам нужны лишь заголовки этих ответов, поэтому мы используем метод HEAD вместо метода GET. Однако вместо новых мест расположения перемещенных ссылок программа печатает отформатированную версию заголовка `Last-Modified`, если он доступен.

Пример 11.6. fresh-links.php

```
require 'HTTP/Request.php';

function pc_link_extractor($s) {
    $a = array();
    if (preg_match_all(
                '/<A\s+.*?HREF=[\"\']?([^\"\' >]*)[\"\']?[^>]*>(.*?)<\/A>/i',
                $s,$matches,PREG_SET_ORDER)) {
        foreach($matches as $match) {
            array_push($a,array($match[1],$match[2]));
        }
    }
```

```
    return $a;
}

$url = $_SERVER['argv'][1];

// извлекаем URL в цикле, чтобы следовать переадресациям
$done = 0;
while (! $done) {
    $req = new HTTP_Request($url);
    $req->sendRequest();
    if ($response_code = $req->getResponseCode()) {
        if ((intval($response_code/100) == 3) &&
            ($location = $req->getResponseHeader('Location'))) {
            $url = $location;
        } else {
            $done = 1;
        }
    } else {
        return false;
    }
}

// определяем базовый url из url
// при этом не обращаем внимания на тег <base> на этой странице
$base_url = preg_replace('{^(.*/)([^/]*)$}','\\1',$req->_url->getURL());

// запоминаем ссылки, которые мы посетили, чтобы не посещать
// их более одного раза
$seen_links = array();

if ($body = $req->getResponseBody()) {
    $links = pc_link_extractor($body);
    foreach ($links as $link) {
        // пропускаем https URL
        if (preg_match('{^https://}',$link[0])) {
            continue;
        }
        // вычисляем относительные ссылки
        if (! (preg_match('{^(http|mailto):}',$link[0]))) {
            $link[0] = $base_url.$link[0];
        }
        // пропускаем данную ссылку, если мы ее уже видели
        if ($seen_links[$link[0]]) {
            continue;
        }

        // отмечаем данную ссылку как просмотренную
        $seen_links[$link[0]] = true;

        // выводим содержимое ссылки, на которой находимся
        print $link[0].': ';
        flush();

        // посещаем ссылку
```

```
        $req2 = new HTTP_Request($link[0],
                                array('method' => HTTP_REQUEST_METHOD_HEAD));
        $now = time();
        $req2->sendRequest();
        $response_code = $req2->getResponseCode();

        // если получение содержимого закончилось успешно
        if ($response_code == 200) {
            // получаем заголовок Last-Modified
            if ($lm = $req2->getResponseHeader('Last-Modified')) {
                $lm_utc = strtotime($lm);
            } else {
                // или устанавливаем Last-Modified в текущее время
                $lm_utc = $now;
            }
            print strftime('%c',$lm_utc);
        } else {
            // в противном случае, выводим код ответа
            print $response_code;
        }
        print "\n";
    }
}
```

12

XML

12.0. Введение

Язык XML приобрел большую популярность как формат для обмена информацией и передачи сообщений. Эта роль XML еще более возросла с развитием и широким распространением веб-сервисов. PHP, при помощи нескольких дополнительных расширений, довольно легко позволяет читать и создавать XML-документы, практически на все случаи жизни.

Теги XML предоставляют разработчикам способ структурированной разметки данных, объединенных в древовидную иерархию. Язык XML можно рассматривать как «CSV на стероидах». Действительно можно использовать XML для обычного хранения записей, состоящих из последовательности полей. Только вместо простого отделения полей друг от друга запятыми в XML можно указать имя поля, тип и атрибуты непосредственно рядом с информацией.

С другой стороны, XML – это язык представления документов. Например, «PHP. Сборник рецептов» была написана с использованием XML. Книга разделена на главы, каждая глава разделена на рецепты, а каждый рецепт состоит из разделов «Задача», «Решение» и «Обсуждение». Содержимое разделов состоит из абзацев, таблиц, рисунков и примеров. Точно так же статья, опубликованная на веб-странице, может быть разделена на название страницы, заголовок, автора фрагмента, собственно повествование, различные боковые панели, сопутствующие ссылки и дополнительное содержание.

Внешне XML очень напоминает HTML. Оба языка используют для разметки текста теги, заключенные в угловые скобки `<` и `>`. Но язык XML одновременно и более строг, и более свободен, чем HTML. Большая строгость XML проявляется в том, что он требует закрытия всех тегов контейнеров. Не допускается открытие элементов без соответствующего закрывающего тега. Свобода XML в том, что он не ограничен использованием конечного списка тегов, таких как `<a>`, `<img>` и `<h1>`. Наоборот,

вы вправе использовать последовательности любых имен тегов, которые наилучшим образом подходят для описания ваших данных.

Следующими ключевыми отличиями XML от HTML являются чувствительность к регистру, обязательное заключение атрибутов в кавычки и обработка пробелов. Если в HTML теги `<B>` и `<b>` это один и тот же тег выделения полужирным шрифтом, то в XML они будут представлять собой два разных тега. В HTML можно зачастую опустить кавычки при указании значений для атрибутов, синтаксис XML требует обязательного указания кавычек в этом случае. Поэтому в XML всегда следует писать:

```
<element attribute="value">
```

Кроме того, анализаторы HTML, как правило, игнорируют пробелы, поэтому ряд из 20 последовательных пробелов трактуется ими как один пробел. Анализаторы XML сохраняют пробельный символ, если только не получают на этот счет других прямых указаний. Поскольку все элементы должны быть закрыты, пустые элементы должны заканчиваться символами />. Например, в HTML конец строки обозначается тегом `<br>`, в то же время в XML он записывается как `<br/>`.[1]

Есть и еще одно важное ограничение, налагаемое на документы XML. Поскольку любой документ XML может быть представлен в виде дерева составляющих его элементов, то самый крайний, верхний в иерархии элемент принято называть корневым (*root*) *элементом*. Подобно тому как дерево имеет один ствол, XML-документ должен иметь единственный корневой элемент. В предыдущем примере с книгой это означает, что главы должны располагаться внутри тега книги. Если нам требуется поместить несколько книг в один документ, то надо поставить их в книжный шкаф или другой аналогичный контейнер. Это ограничение касается только корня документа. Повторим, что точно так же, как дерево может иметь несколько ветвей, растущих из одного ствола, допустимо хранить несколько книг в одном книжном шкафу.

Эта глава не ставит своей целью научить вас XML. Для получения начальных сведений об XML стоит обратиться к книге Эрика Рэя (Erik T. Ray) «Learning XML».[2] Основательным руководством по всем аспектам XML является книга «XML in a Nutshell» Элиота Расти Гарольда (Elliotte Rusty Harold) и Скотта Минса (W. Scott Means). Обе книги выпущены в издательстве «O'Reilly & Associates».[3]

Теперь, после краткого знакомства с основными правилами, приведем небольшой пример: если вы библиотекарь и хотите перевести ваш бу-

---

[1] Именно поэтому функция `nl2br()` выводит `<br />`; ее вывод совместим с XML.

[2] Рэй Э. «Изучаем XML». – Пер. с англ. – СПб: Символ-Плюс, 2001.

[3] Гарольд Э., Минс С. «XML. Справочник». – Пер. с англ. – СПб: Символ-Плюс, 2002.

мажный карточный каталог в XML-представление, начните с такого базового множества тегов:

```
<book>
    <title>PHP Cookbook</title>
    <author>Sklar, David and Trachtenberg, Adam</author>
    <subject>PHP</subject>
</book>
```

С этого момента можно добавлять новые элементы или модифицировать уже существующие. Например, элемент `<author>` можно разделить на имя и фамилию, или разрешить для него множественные записи, так чтобы два автора не размещались в одном поле.

Первые три рецепта этой главы посвящены созданию и чтению XML-документов. Рецепт 12.1 показывает, как создавать XML без применения дополнительных инструментов. В рецепте 12.2 объясняется использование расширения DOM XML для создания XML-документов в стандартной манере. Чтение XML с применением DOM будет предметом рецепта 12.3.

Но на одном голом XML дело не заканчивается. После того как весь XML-документ собран, возникает реальный вопрос: «А что с этим делать дальше?» При помощи анализатора, основанного на событиях, как это описано в рецепте 12.4, можно предпринимать действия в соответствии с тегами элементов. Например, сохранять данные в легко обрабатываемых структурах или изменять форматирование текста.

С помощью XSLT можно, используя страницу стиля, преобразовать XML-документ в удобный для просмотра вывод.[1] Отделяя таким образом содержание документа от его представления, можно создать одну страницу стиля для веб-броузеров, другую для PDA, третью для сотовых телефонов, и все это без изменения собственно содержимого документа. Такие преобразования являются предметом рецепта 12.5.

Применяя такие протоколы, как XML-RPC или SOAP, можно организовать обмен сообщениями между вашим персональным компьютером и сервером, или, наоборот, самому поработать в качестве сервера. Таким способом вы сможете разместить свою картотеку в Интернете и дать возможность другим пользователям запрашивать ее и извлекать записи об отдельных книгах в том формате, который подходит для анализа и отображения этих данных в их приложениях. Другим применением могло бы стать создание RSS-рассылки, обновляемой всякий раз, когда в библиотеку поступает новая книга. Клиенты и серве-

[1] XSLT, или XML Style Language Transformations, – язык преобразований, основанный на языке стилей XML. Но XSLT – это не просто этап форматирования, он представляет собой инструмент XML-обработки и применяется как для преобразования одной разновидности XML в другую, так и для преобразования XML в другой формат, например в HTML или обычный текст. – *Примеч. науч. ред.*

ры XML-RPC рассматриваются в рецептах 12.6 и 12.7 соответственно. Рецепты 12.8 и 12.9 посвящены клиентам и серверам на основе протокола SOAP. WDDX, формат обмена информацией, порожденный языком ColdFusion, станет темой рецепта 12.10. Чтению RSS-рассылок, популярному формату синдикации заголовков, основанному на XML, посвящен рецепт 12.11.

Как и многие передовые технологии, некоторые инструменты PHP для работы с XML не обладают пока всей полнотой возможностей и не застрахованы от ошибок. Сейчас XML – область активной разработки в PHP-сообществе, добавление в него новых возможностей и исправление ошибок происходит постоянно, на регулярной основе. Вследствие этого многие XML-функции, описанные и документированные в этой книге, рассматриваются как экспериментальные. Иногда это означает, что функция готова на 99%, но может содержать некоторое количество небольших ошибок. В другом случае это означает, что имя и само поведение функции могут быть полностью изменены. Если функция находится в очень нестабильном состоянии, мы отдельно упоминаем об этом в рецепте.

И все же мы постарались документировать эти функции, т. к. в настоящее время планируется их включение в версию PHP 4.3. Поскольку работа с XML превращается в важную область программирования, не имеет смысла исключать эти рецепты из книги. Кроме того, мы хотели обеспечить использование самых новых функций в наших примерах. Это, однако, может привести к трудностям, если имена и прототипы функций изменятся. Обнаружив, что рецепт не работает так, как вы ожидали, пожалуйста, обратитесь к онлайн-версии руководства по PHP или к разделу *Errata* на странице *http://www.oreilly.com/catalog/phpckbk*.

12.1. Генерация XML вручную

Задача

Необходимо сгенерировать XML-документ. Например, требуется предоставить XML-версию данных для анализа другой программе.

Решение

Надо в цикле пройти по всем данным и вывести их, заключив в соответствующие теги XML:

```
header('Content-Type: text/xml');
print '<?xml version="1.0"?>' . "\n";
print "<shows>\n";

$shows = array(array('name'     => 'Simpsons',
                     'channel'  => 'FOX',
```

```
                        'start'    => '8:00 PM',
                        'duration' => '30'),

                array('name'     => 'Law & Order',
                        'channel'  => 'NBC',
                        'start'    => '8:00 PM',
                        'duration' => '60'));

foreach ($shows as $show) {
    print "    <show>\n";
    foreach($show as $tag => $data) {
        print "        <$tag>" . htmlspecialchars($data) . "</$tag>\n";
    }
    print "    </show>\n";
}

print "</shows>\n";
```

Обсуждение

Вывод XML вручную требует множества вложенных циклов `foreach`, поскольку выполняются итерации по массивам. Кроме того, здесь есть несколько хитрых нюансов. Во-первых, необходимо вызвать функцию `header()`, указать корректный заголовок `Content-Type` для нашего документа. Наша программа отправляет данные в формате XML, а не в HTML, поэтому заголовок должен указывать тип содержимого как `text/xml`.

Далее, в зависимости от значения параметра настройки `short_open_tag`, попытка напечатать объявление XML может непроизвольно включить обработку PHP. Символы `<?` элемента `<?xml version="1.0"?>` совпадают с краткой формой открывающего тега PHP-кода. Для вывода объявления в броузере необходимо либо запретить в конфигурации краткую форму тега, либо выводить эти строки из PHP. В нашем «Решении» мы выбрали последний из этих вариантов.

И наконец, сами элементы должны быть превращены в escape-последовательности. Например, символ `&` при показе строки `Law & Order` должен быть выдан в виде `&`. Данные преобразуются в escape-последовательности посредством функции `htmlspecialchars()`.

Результат работы примера, приведенного в разделе «Решение», выглядит так:

```
<?xml version="1.0"?>
<shows>
    <show>
        <name>Simpsons</name>
        <channel>FOX</channel>
        <start>8:00 PM</start>
        <duration>30</duration>
    </show>
    <show>
        <name>Law &amp; Order</name>
```

```
        <channel>NBC</channel>
        <start>8:00 PM</start>
        <duration>60</duration>
    </show>
</shows>
```

См. также

Рецепт 12.2 о том, как генерировать XML с применением DOM; рецепт 12.3 о чтении XML с помощью DOM; документацию по функции `htmlspecialchars()` на *http://www.php.net/htmlspecialchars*.

12.2. Генерация XML с применением DOM

Задача

Необходимо сгенерировать XML, используя более высокий уровень организации данных, чем в случае применения операторов вывода (`print`) и циклов.

Решение

Используйте расширение PHP DOM XML для создания и заполнения соответствующего DOM-объекта, затем вызовите функцию `dump_mem()` или `dump_file()` для того, чтобы сгенерировать корректный (well-formed) XML-документ:

```
// создаем новый объект - документ
$dom = domxml_new_doc('1.0');

// создаем корневой элемент, <book>, и добавляем его в документ
$book = $dom->append_child($dom->create_element('book'));

// создаем элемент title и присоединяем его к переменной-элементу $book
$title = $book->append_child($dom->create_element('title'));

// создаем текст и атрибут cover для элемента $title
$title->append_child($dom->create_text_node('PHP Cookbook'));
$title->set_attribute('cover', 'soft');

// создаем и добавляем элементы author в переменную-элемент $book
$sklar = $book->append_child($dom->create_element('author'));

// создаем и добавляем текст для элемента author
$sklar->append_child($dom->create_text_node('Sklar'));

// повторяем эти действия для второго элемента authors
$trachtenberg = $book->append_child($dom->create_element('author'));
$trachtenberg->append_child($dom->create_text_node('Trachtenberg'));

// печатаем полностью отформатированную версию документа DOM в форме XML
echo $dom->dump_mem(true);
```

```
Результат работы примера:

<?xml version="1.0"?>
<book>
  <title cover="soft">PHP Cookbook</title>
  <author>Sklar</author>
  <author>Trachtenberg</author>
</book>
```

Обсуждение

Одиночный компонент документа называется *узлом*. Существует масса различных типов узлов, из которых чаще других используются три – элемент, атрибут и текст. Рассмотрим пример:

```
<book cover="soft">PHP Cookbook</book>
```

С позиции расширения DOM XML, элемент `book` имеет тип `XML_ELEMENT_NODE`, пара «имя-значение» `cover="soft"` относится к типу `XML_ATTRIBUTE_NODE`, а строка `PHP Cookbook` – к типу `XML_TEXT_NODE`.[1]

Для DOM-анализа PHP использует библиотеку libxml, разработанную для проекта Gnome. Ее можно загрузить с *http://www.xmlsoft.org*. Чтобы подключить и активировать эту библиотеку, необходимо сконфигурировать PHP с параметром `--with-dom`.[2]

Методы DOM XML в версии PHP 4.3 применяются согласно следующему шаблону. Вы создаете объект как узел-элемент или как узел-текст, а затем помещаете его в то место (узел типа элемент) дерева, где он должен находиться.

Перед тем как создавать элементы дерева, необходимо создать сам документ, передав конструктору объекта единственный параметр – версию XML:

```
$dom = domxml_new_doc('1.0');
```

Теперь можно создавать новые элементы, принадлежащие документу. Несмотря на то что все узлы-элементы имеют смысл лишь внутри до-

---

[1] Здесь прослеживается прямая аналогия с базовыми компонентами языка XML. Ключевым компонентом XML является элемент (изолированная с помощью тегов область данных). Элементы могут разбиваться на подэлементы и выполняют роль связывания имен-меток и других свойств с данными. Кроме того, элементы могут включать пары «имя-значение», именуемые атрибутами. – *Примеч. науч. ред.*

[2] При работе в ОС Windows надо скопировать соответствующую библиотеку (файл с расширением *.DLL*) из подкаталога *\DLLS* того каталога, в который установлен PHP, в подкаталог *\SYSTEM32* каталога %Windows%. Для пользователей PHP версии не выше 4.2 это библиотека *libxml2.dll*, а для пользователей версии не ниже 4.3.0 – библиотека *iconv.dll*. – *Примеч. науч. ред.*

кумента, операция присоединения (подключения) узла к документу должна быть выполнена явным образом:

```
$book_element = $dom->create_element('book');
$book = $dom->append_child($book_element);
```

Здесь мы создаем новый элемент book и назначаем его объекту $book_element. Для того чтобы создать корневой элемент документа, мы должны добавить наш новый объект (в качестве дочернего) к объекту-документу $dom. Результатом работы метода append_child() будет новый объект, обладающий местоположением (включенный в документ) в объекте DOM.

Все узлы создаются с помощью вызова метода объекта $dom. После создания узел может быть добавлен к любому элементу дерева. Элемент, чей метод append_child() мы вызываем, определяет местоположение узла в дереве. В предыдущем случае $book_element добавлялся к $dom. Элемент, добавленный к $dom, представляет узел верхнего уровня документа, или *корневой узел*.

Можно также добавить новый дочерний элемент в $book. Поскольку $book представляет собой дочерний элемент документа $dom, новый элемент по аналогии будет внуком документа $dom:

```
$title_element = $dom->create_element('title');
$title = $book->append_child($title_element);
```

С помощью вызова $book->append_child() этот фрагмент кода помещает элемент $title_element под элементом $book.

Для добавления текста внутрь тегов <title></title> создайте текстовый узел с помощью функции create_text_node() и добавьте его к $title:

```
$text_node = $dom->create_text_node('PHP Cookbook');
$title->append_child($text_node);
```

Элемент $title уже добавлен к документу, поэтому нет необходимости добавлять его в $book повторно.

Порядок, в котором дочерние элементы добавляются к узлам, не важен. Следующие четыре строчки, которые сначала добавляют текстовый узел к $title_element, а затем к $book, эквивалентны предыдущему коду:

```
$title_element = $dom->create_element('title');
$text_node = $dom->create_text_node('PHP Cookbook');

$title_element->append_child($text_node);
$book->append_child($title_element);
```

Для добавления атрибута надо вызвать метод set_attribute() нужного узла, передав имя атрибута и его значение в качестве аргументов:

```
$title->set_attribute('cover', 'soft');
```

Если теперь напечатать элемент title, он будет выглядеть примерно так:

```
<title cover="soft">PHP Cookbook</title>
```

Теперь можно вывести весь документ в виде строки или в файл:

```
// помещаем строковое представление XML-документа в $books
$books = $dom->dump_mem();

// записываем XML-документ в файл books.xml
$dom->dump_file('books.xml', false, true);
```

Единственный параметр, который принимает метод `dump_mem()`, — это необязательное логическое значение. Пустое значение (или `false`) означает «вернуть результат как одну длинную строку». Значение `true` порождает удобно отформатированный документ XML, в котором дочерние узлы выводятся с соответствующими отступами, например:

```
<?xml version="1.0"?>
<book>
  <title cover="soft">PHP Cookbook</title>
</book>
```

Методу `dump_file()` можно передавать до трех значений. Первое обязательное значение — это имя файла. Второе значение определяет, должен ли файл сжиматься с помощью *gzip*. Последнее значение представляет ту же самую опцию форматирования, что и в методе `dump_mem()`.

См. также

Рецепт 12.1 о составлении документов XML без использования DOM; рецепт 12.3 об анализе документов XML с помощью DOM; документацию по функции `domxml_new_dom()` на *http://www.php.net/domxml-new-dom* и общее описание функций DOM на *http://www.php.net/domxml*; более подробно о базовой C-библиотеке DOM на *http://xmlsoft.org/*.

12.3. Анализ XML с помощью DOM

Задача

Необходимо проанализировать XML-файл с помощью DOM API. Он преобразует файл в дерево, которое можно обработать, применяя функции DOM. DOM позволяет без труда найти и извлечь элементы, удовлетворяющие определенному набору критериев.

Решение

Для этого надо прибегнуть к PHP-расширению DOM XML. Ниже показано, как читать XML из файла:

```
$dom = domxml_open_file('books.xml');
```

Теперь приведем пример чтения XML из переменной:

```
$dom = domxml_open_mem($books);
```

Можно также извлекать один конкретный узел. Следующее выражение демонстрирует чтение корневого узла:

```
$root = $dom->document_element();
```

Приведем пример обработки всех узлов документа с помощью рекурсии типа «сначала вглубь»:

```
function process_node($node) {
    if ($node->has_child_nodes()) {
        foreach($node->child_nodes() as $n) {
            process_node($n);
        }
    }

    // обработка элементов нижнего уровня
    if ($node->node_type() == XML_TEXT_NODE) {
        $content = rtrim($node->node_value());
        if (!empty($content)) {
            print "$content\n";
        }
    }

}
process_node($root);
```

Обсуждение

DOM W3C предоставляет независимую от платформы и языка технологию описания структуры и содержимого документа. DOM позволяет прочитать документ XML в дерево узлов, а затем перемещаться по дереву для поиска информации об определенном элементе или группе элементов, удовлетворяющей критерию поиска. Подобный способ называется анализом *на основе дерева*. В отличие от него, функции XML, не основанные на DOM, позволяют проводить анализ на основе событий.

Кроме того, можно модифицировать структуру документа путем создания, редактирования и удаления узлов. Фактически можно использовать функции DOM XML для создания нового XML-документа (и его элементов) с нуля; см. рецепт 12.2.

Одним из главных преимуществ DOM является то, что, следуя спецификации W3C, функции DOM во многих языках реализуются одинаковым образом. Поэтому работа по переносу логики и инструкций из одного приложения в другое значительно упрощается. Версия PHP 4.3 поставляется с обновленным комплектом функций, более строго согласованным со стандартом DOM, чем в предыдущих версиях PHP. Однако эти функции согласованы еще не на 100%. Будущие версии PHP должны будут обеспечить более точное совмещение. При этом некоторые приложения могут оказаться неработоспособными и потребовать незначительной доработки. Чтобы больше узнать о производимых изменениях, обратитесь к материалам по DOM XML онлайн-версии руко-

водства по PHP по адресу *http://www.php.net/domxml*. Функции более ранних версий PHP по-прежнему доступны, но их применение очень не одобряется.

Модель DOM большая и сложная. Более подробную информацию ищите в спецификации на *http://www.w3.org/DOM/*, книге «XML in a Nutshell» и на форумах, посвященных DOM.

Для DOM-анализа в PHP используется модуль *libxml*, разработанный для проекта Gnome. Загрузить его можно с адреса *http://www.xmlsoft.org*. Для подключения этого модуля сконфигурируйте PHP с опцией `--with-dom`.

Функции DOM в PHP объектно-ориентированны. Чтобы перемещаться от узла к узлу, вызывайте такие методы, как `$node->child_nodes()`, который возвращает дочерние объекты-узлы, и `$node->parent_node()`, возвращающий объект родительского узла. При обработке узла проверьте его тип и вызовите соответствующий метод:

```
// $node - это проанализированный с помощью DOM
// узел <book cover="soft">PHP Cookbook</book>
$type = $node->node_type();

switch($type) {
case XML_ELEMENT_NODE:
    // Я - тег. У меня есть свойство «имя тега».
    print $node->node_name();  // печатает свойство имя тега: "book"
    print $node->node_value(); // null
    break;
case XML_ATTRIBUTE_NODE:
    // Я - атрибут. У меня есть пара свойств «имя-значение».
    print $node->node_name();  // печатает свойство имя: "cover"
    print $node->node_value(); // печатает свойство значение: "soft"
    break;
case XML_TEXT_NODE:
    // Я - часть текста внутри элемента.
    // У меня есть свойства имя и содержание.
    print $node->node_name();  // печатает свойство имя: "#text"
    print $node->node_value(); // печатает свойство содержание:
                               // "PHP Cookbook"
    break;
default:
    // другой тип
    break;
}
```

Для автоматического поиска определенного элемента в дереве DOM используйте функцию `get_elements_by_tagname()`. Ниже показано, как это сделать в случае нескольких записей book:

```
<books>
    <book>
        <title>PHP Cookbook</title>
        <author>Sklar</author>
```

```
        <author>Trachtenberg</author>
        <subject>PHP</subject>
    </book>
    <book>
        <title>Perl Cookbook</title>
        <author>Christiansen</author>
        <author>Torkington</author>
        <subject>Perl</subject>
    </book>
</books>
```

Следующий фрагмент демонстрирует, как найти всех авторов:

```
// находим и печатаем всех авторов
$authors = $dom->get_elements_by_tagname('author');

// выполняем цикл по элементам автор
foreach ($authors as $author) {
    // метод child_nodes() извлекает значения элементов автор
    $text_nodes = $author->child_nodes();

    foreach ($text_nodes as $text) {
         print $text->node_value();
    }
    print "\n";
}
```

Функция `get_elements_by_tagname()` возвращает массив объектов узлов указанного элемента. Выполняя цикл по каждому потомку элемента, можно извлечь текстовый узел, связанный с этим элементом. Теперь можно получить значения узлов, которыми в данном случае являются имена авторов книг, например `Sklar` и `Trachtenberg`.

См. также

Рецепт 12.1 о составлении XML-документов без использования DOM; рецепт 12.2 о составлении XML-документов с использованием DOM; рецепт 12.4 об анализе XML на основе событий; документацию на функции `domxml_open_file()` (по адресу *http://www.php.net/domxml-open-file*), `domxml_open_mem()` (по адресу *http://www.php.net/*), *domxml-open-mem* и обобщенную информацию по функциям DOM по адресу *http://www.php.net/domxml*; более подробные сведения о базовой C-библиотеке DOM можно получить по адресу *http://xmlsoft.org/*.

12.4. Анализ XML с помощью SAX

Задача

Необходимо проанализировать XML-документ и отформатировать его на основе событий. Например, когда анализатор встречается с новым открывающим или закрывающим тегом элемента. Допустим, требуется преобразовать RSS-рассылку в HTML.

Решение

Используйте анализирующие функции XML-расширения PHP:

```
$xml = xml_parser_create();
$obj = new Parser_Object;  // класс, принимающий участие в анализе

xml_set_object($xml,$obj);
xml_set_element_handler($xml, 'start_element', 'end_element');
xml_set_character_data_handler($xml, 'character_data');
xml_parser_set_option($xml, XML_OPTION_CASE_FOLDING, false);

$fp = fopen('data.xml', 'r')
        or die("Can't read XML data.");
while ($data = fread($fp, 4096)) {
  xml_parse($xml, $data, feof($fp))
    or die("Can't parse XML data");
}
fclose($fp);

xml_parser_free($xml);
```

Обсуждение

Анализирующим XML-функциям для работы требуется библиотека *expat*. Но поскольку версии веб-сервера Apache 1.3.7 и выше комплектуются библиотекой *expat*, эта библиотека уже установлена на большинстве машин. Поскольку в PHP эти функции доступны по умолчанию, нет необходимости явным образом конфигурировать поддержку XML в PHP.

Библиотека *expat* анализирует XML-документы и позволяет настроить анализатор таким образом, чтобы он вызывал нужные функции при столкновении с различными частями файла, такими как открывающий или закрывающий тег элемента или символьные данные (текст внутри тегов). Основываясь на имени тега, можно выбирать между форматированием или игнорированием данных. Это методика известна как анализ *на основе событий* и противостоит DOM XML, использующей анализатор на основе дерева.

Популярным видом API для анализа XML на основе событий является SAX (Simple API for XML – простой API для XML). Разработанный изначально для Java, SAX получил распространение и в других языках. Функции XML в PHP поддерживают принятые в SAX соглашения. Более подробную информацию о последней версии SAX – SAX2 см. в книге Дэвида Браунела (David Brownell) «SAX2», O'Reilly.

PHP поддерживает два интерфейса для библиотеки *expat*: процедурный и объектно-ориентированный. Поскольку процедурный интерфейс вынуждает использовать глобальные переменные для выполнения сколько-нибудь существенной задачи, мы предпочитаем его объектно-ориентированную версию. С помощью объектно-ориентированного ин-

терфейса можно связать объект с анализатором и взаимодействовать с объектом во время обработки XML-документа. Такой подход позволяет использовать собственные свойства объекта вместо глобальных переменных.

Ниже приведен пример использования данной библиотеки, показывающий, как обрабатывать RSS-рассылку и преобразовывать ее в HTML. Более подробную информацию о RSS см. в рецепте 12.11. Сценарий начинается со стандартного кода обработки XML, следующего за созданием объекта для специального анализа RSS:

```
$xml = xml_parser_create();
$rss = new pc_RSS_parser;

xml_set_object($xml, $rss);
xml_set_element_handler($xml, 'start_element', 'end_element');
xml_set_character_data_handler($xml, 'character_data');
xml_parser_set_option($xml, XML_OPTION_CASE_FOLDING, false);

$feed = 'http://pear.php.net/rss.php';
$fp = fopen($feed, 'r')
            or die("Can't read RSS data.");

while ($data = fread($fp, 4096)) {
  xml_parse($xml, $data, feof($fp))
            or die("Can't parse RSS data");
}
fclose($fp);

xml_parser_free($xml);
```

После создания нового анализатора XML и экземпляра класса `pc_RSS_parser` задается конфигурация анализатора. Для начала объект связывается с анализатором; это действие дает указание анализатору вызывать методы объекта, а не глобальные функции. Затем вызываются функции для указания имен методов `xml_set_element_handler()` и `xml_set_character_data_handler()`, которые анализатор должен вызывать, когда встречает элементы и символьные данные. Первым аргументом обеих функций является экземпляр анализатора, остальные аргументы – это имена самих вызываемых функций. В случае функции `xml_set_element_handler()` второй и третий аргументы – это функции, которые должны быть вызваны, соответственно, при встрече открывающего и закрывающего тегов. Функция `xml_set_character_data_handler()` принимает только один дополнительный аргумент – имя функции, которую нужно вызвать для обработки символьных данных.

Поскольку с нашим анализатором был связан объект, то в момент, когда анализатор обнаруживает строку `<tag>data</tag>`, он вызывает метод `$rss->start_element()` при достижении тега `<tag>`, метод `$rss->character_data()` – при достижении `data` и метод `$rss->end_element()` – при достижении тега `</tag>`. Анализатор нельзя сконфигурировать для автоматического вызова уникального метода для каждого конкретного

тега – вы должны организовать обработку различных тегов самостоятельно. Так, пакет PEAR `XML_Transform` предоставляет простой способ назначения обработчиков на основе тегов.

Последняя опция конфигурации анализатора XML запрещает автоматическое преобразование анализатором всех тегов в верхний регистр. По умолчанию анализатор записывает теги большими буквами, поэтому теги `<tag>` и `<TAG>` становятся одним и тем же элементом. Так как XML чувствителен к значению регистра, а большинство рассылок используют имена в нижнем регистре, эта функциональная возможность должна быть принудительно запрещена.

После завершения конфигурирования анализатора ему передаются данные:

```
$feed = 'http://pear.php.net/rss.php';
$fp = fopen($feed, 'r')
            or die("Can't read RSS data.");

while ($data = fread($fp, 4096)) {
  xml_parse($xml, $data, feof($fp))
            or die("Can't parse RSS data");
}

fclose($fp);
```

Для того чтобы сдержать рост используемой памяти, обрабатываемый файл загружается порциями по 4096 байт и передается анализатору по одному блоку за раз. Для этого вам потребуется написать функции-обработчики, которые будут размещать текст, поступающий от нескольких вызовов, учитывая, что за один раз вся строка не будет доставлена.

Хотя PHP и прекращает работу всех анализаторов при завершении запроса, можно закрыть анализатор вручную с помощью вызова функции `xml_parser_free()`.

Теперь, когда основной процесс анализа определен, добавьте классы `pc_RSS_item` и `pc_RSS_parser` для обработки RSS-документа, как показано в примерах 12.1 и 12.2.

Пример 12.1. pc_RSS_item

```
class pc_RSS_item {

  var $title = '';
  var $description = '';
  var $link = '';

  function display() {
    printf('<p><a href="%s">%s</a><br />%s</p>',
            $this->link,htmlspecialchars($this->title),
            htmlspecialchars($this->description));
  }
}
```

Пример 12.2. pc_RSS_parser

```
class pc_RSS_parser {

  var $tag;
  var $item;

  function start_element($parser, $tag, $attributes) {
    if ('item' == $tag) {
      $this->item = new pc_RSS_item;
    } elseif (!empty($this->item)) {
      $this->tag = $tag;
    }
  }

  function end_element($parser, $tag) {
    if ('item' == $tag) {
      $this->item->display();
      unset($this->item);
    }
  }

  function character_data($parser, $data) {
    if (!empty($this->item)) {
      if (isset($this->item->{$this->tag})) {
        $this->item->{$this->tag} .= trim($data);
      }
    }
  }
}
```

Класс `pc_RSS_item` обеспечивает интерфейс доступа к отдельному экземпляру рассылки. Он скрывает детали отображения отдельного экземпляра от общего процесса анализа и облегчает переустановку данных для нового экземпляра с помощью вызова функции `unset()`.

Метод `pc_RSS_item::display()` отвечает за вывод экземпляра RSS в формате HTML. Он вызывает функцию `htmlspecialchars()` для повторного кодирования всех нуждающихся в этом элементов, поскольку библиотека *expat* декодирует их в процессе анализа документа в обычные символы. Однако такая перекодировка повреждает рассылки, которые вместо простого текста размещают в названии и описании HTML-элементы.

Метод `start_element()`, класса `pc_RSS_parser()`, принимает три параметра: анализатор XML, имя тега и массив пар атрибут-значение (если таковые имеются) элемента. PHP автоматически применяет эти значения для обработчика, как часть процесса анализа.

Метод `start_element()` проверяет значение переменной `$tag`. Если оно равно `item`, следовательно, анализатор нашел новый экземпляр RSS и необходимо создать новый экземпляр класса `pc_RSS_item`. В противном случае с помощью функции `empty()` проверяется, пустое или нет значе-

ние `$this->item`. Если оно не пусто, то анализатор находится внутри элемента `item`. В этом случае необходимо записать имя тега, чтобы метод `character_data()` знал, какому свойству присвоить его значение. Если значение `$this->item` пустое, то эта часть RSS-рассылки считается ненужной и игнорируется.

Когда анализатор находит закрывающий тег `item`, то соответствующий метод `end_element()` сначала печатает элемент RSS, а затем завершает работу после удаления объекта.

В заключение метод `character_data()` отвечает за присваивание значений `title`, `description` и `link` элементу RSS. Убедившись, что анализатор находится внутри элемента `item`, он проверяет, является ли текущий тег одним из свойств объекта `pc_RSS_item`. Без этой проверки, если бы анализатор встретил элемент, отличный от перечисленных выше трех элементов, его значение также было бы назначено объекту. Фигурные скобки {} необходимы для указания порядка разыменования свойства объекта. Обратите внимание, что вместо прямого присваивания к свойству добавляется результат операции усечения пробелов – `trim($data)`. Это делается для того, чтобы учесть случаи, в которых символьные данные, извлекаемые методом `fread()`, поступают блоками по 4096 байт. Это также позволяет удалить окружающие пробельные символы, найденные в RSS-рассылке.

Если запустить приведенный код для простой RSS-рассылки вида:

```
<?xml version="1.0"?>
<rss version="0.93">
<channel>
  <title>PHP Announcements</title>
  <link>http://www.php.net/</link>
  <description>All the latest information on PHP.</description>

  <item>
    <title>PHP 5.0 Released!</title>
    <link>http://www.php.net/downloads.php</link>
    <description>The newest version of PHP is now available.</description>
  </item>
</channel>
</rss>
```

то на выходе мы получим следующий HTML-текст:

```
<p><a href="http://www.php.net/downloads.php">PHP 5.0 Released!</a><br />
The newest version of PHP is now available.</p>
```

См. также

Рецепт 12.3 об основанном на дереве анализе XML при помощи DOM; более подробную информацию об анализе RSS в рецепте 12.11; документацию по функции `xml_parser_create()` на *http://www.php.net/xml-parser-create*, по функции `xml_element_handler()` на *http://www.php.net/*

xml-element-handler, по функции `xml_character_handler()` на *http://www.php.net/xml-character-handler*, по функции `xml_parse()` на *http://www.php.net/xml-parse* и обобщенную информацию о функциях XML по адресу *http://www.php.net/xml*; официальный сайт SAX, расположенный по адресу *http://www.saxproject.org/*.

12.5. Преобразование XML с помощью XSLT

Задача

Есть XML-документ и таблица стилей XSL. Необходимо преобразовать документ с помощью XSLT и представить результаты. Это позволит применять таблицы стилей к данным и создавать различные версии содержания для представления в разных средствах информации.

Решение

Используйте расширение PHP XSLT:

```
$xml = 'data.xml';
$xsl = 'stylesheet.xsl';

$xslt = xslt_create();
$results = xslt_process($xslt, $xml, $xsl);

if (!$results) {
    error_log("XSLT Error: #".xslt_errno($xslt).": ".xslt_error($xslt));
}

xslt_free($xslt);
```

Преобразованный текст сохраняется в переменной `$results`.

Обсуждение

XML-документы описывают содержание данных, но в них нет никакой информации о том, как отображать эти данные. Однако если содержание документа связано с таблицей стилей, описываемой с помощью XSL (eXtensible Stylesheet Language – открытый язык таблиц стилей), то это содержание будет отображаться согласно определенным правилам визуализации.

Связующим звеном между XML и XSL является XSLT, который определяет преобразования открытого языка таблиц стилей. Эти преобразования применяют к XML-данным наборы правил, перенумерованные в таблице стилей. Поэтому точно так же, как PHP анализирует программу и объединяет ее с пользовательским вводом при создании динамической страницы, XSLT-программа использует XSL и XML для вывода новой страницы, дополнительно содержащей XML, HTML или какой-либо другой формат, который вы можете описать.

Сейчас существует небольшое количество XSLT-программ с различными возможностями и ограничениями. PHP в настоящее время поддерживает лишь XSLT-процессор Sablotron. В будущем станут доступны и другие программы, такие как Xalan и Libxslt. Sablotron можно загрузить с адреса *http://www.gingerall.com*. Чтобы разрешить использование процессора Sablotron для XSLT-обработки, в конфигурации PHP нужно указать две опции: `--enable-xslt` и `--with-xslt-sablot`.

Обработка документов проводится в несколько этапов. Во-первых, необходимо захватить обработчик нового экземпляра XSLT-процессора с помощью функции `xslt_create()`. Затем для преобразования файлов и проверки результатов используйте функцию `xslt_process()`:

```
$xml = 'data.xml';
$xsl = 'stylesheet.xsl';

$xslt = xslt_create();
$results = xslt_process($xslt, $xml, $xsl);
```

Мы начинаем с определения переменных для хранения имен файлов, содержащих XML-данные и таблицу стилей XSL. Эти переменные будут выступать в качестве второго и третьего параметров функции преобразования `xslt_process()`. Если, как в данном случае, четвертый элемент опущен или установлен в значение `NULL`, функция вернет нам результаты преобразования. В противном случае она сохранит результаты преобразования в файле с указанным именем:

```
xslt_process($xslt, $xml, $xsl, 'data.html');
```

Если вы хотите извлекать данные XML и XSL из переменных, а не из файлов, используйте функцию `xslt_process()` с указанием пятого параметра, который позволяет заменить файлы строковыми переменными-заполнителями:

```
// извлекаем информацию из базы
$r = mysql_query("SELECT pages.page AS xml, templates.template AS xsl
                  FROM pages, templates
                  WHERE pages.id=$id AND templates.id=pages.template")
     or die("$php_errormsg");

$obj = mysql_fetch_object($r);
$xml = $obj->xml;
$xsl = $obj->xsl;

// преобразовываем строки в массив args
$args = array('/_xml' => $xml, '/_xsl' => $xsl);

$results = xslt_process($xslt, 'arg:/_xml', 'arg:/_xsl', NULL, $args);
```

При чтении и записи файлов Sablotron поддерживает два типа URI.[1] В PHP значением по умолчанию является «`file:`», поэтому Sablotron

[1] URI (Uniform Resource Identifiers) – единый, или универсальный идентификатор ресурсов. – *Примеч. науч. ред.*

ищет данные в файловой системе. Sablotron также может использовать пользовательский URI «arg:», который разрешает в качестве альтернативы передавать данные с помощью аргументов. Именно эта возможность используется в нашем примере.

В предыдущем примере данные для XML и XSL поступают из базы данных, но точно так же они могут быть получены и из других источников, например могут быть загружены с удаленного ресурса (URL) или получены в результате передачи методом POST. При получении данных создается массив $args. Таким образом, устанавливается связь между именами аргументов и именами переменных. Ключи ассоциативного массива являются именами аргументов, переданных функции xslt_process(); значения аргументов – переменные, содержащие данные. По соглашению именами аргументов являются /_xml и /_xsl, однако можно использовать и другие имена.

Затем вызывается функция xslt_process() и вместо имени файла data.xml используется arg:/_xml со строкой arg:, которая указывает расширению, что нужно заглянуть в массив $args. Так как массив $args передается в качестве пятого параметра, то в качестве четвертого параметра необходимо передать NULL; это заставит функцию вернуть результаты.

Проверка наличия ошибок выполняется с помощью функций xslt_error() и xslt_errno():

```
if (!$results) {
    error_log('XSLT Error: #' . xslt_errno($xslt) . ': ' . xslt_error($xslt));
}
```

Функция xslt_error() возвращает форматированное сообщение, описывающее ошибку, функция xslt_errno() – числовой код ошибки.

Чтобы установить свою собственную программу обработки ошибок, зарегистрируйте нужную функцию с помощью xslt_set_error_handler(). В случае возникновения ошибок эта функция будет автоматически вызываться вместо любого встроенного обработчика.

```
function xslt_error_handler($processor, $level, $number, $messages) {
    error_log("XSLT Error: #$level");
}

xslt_set_error_handler($xslt, 'xslt_error_handler');
```

Хотя PHP прекращает работу любого открытого XSLT-процессора по завершении запроса, ниже показано, как вручную закрыть процессор и освободить занимаемую им память:

```
xslt_close($xslt);
```

См. также

Документацию по функции xslt_create() на *http://www.php.net/xslt-create*, по функции xslt_process() на *http://www.php.net/xslt-process*, по функции xslt_errno() на *http://www.php.net/xslt-errno*, по функции

`xslt_error()` на *http://www.php.net/xslt-error*, по функции `xslt_error_handler()` на *http://www.php.net/xslt-error-handler* и по функции `xslt_free()` на *http://www.php.net/xslt-free*; книгу «*XSLT*» Дуга Тидвела (Doug Tidwell) (издательство O'Reilly).

12.6. Посылка запросов XML-RPC

Задача

Вы хотите быть XML-RPC-клиентом и посылать запросы на сервер. XML-RPC позволяет PHP осуществлять вызовы функций на веб-сервере, даже если они не используют PHP. Затем полученные данные автоматически конвертируются в переменные PHP для использования в вашем приложении.

Решение

Используйте встроенное в PHP расширение XML-RPC с некоторыми вспомогательными функциями. Что касается версии PHP 4.1, то она поставляется с расширением *xmlrpc-epi*. К сожалению, расширение *xmlrpc-epi* не имеет «родных» C-функций, принимающих форматированную строку XML-RPC и посылающих запрос. Однако в более позднее семейство xmlrpc-epi включен набор вспомогательных функций, написанных на PHP и доступных для загрузки с адреса *http://xmlrpc-epi.sourceforge.net/*. Единственным используемым в данном случае файлом является файл *utils.php*, который располагается в каталоге *sample/utils*. Для инсталляции этого файла просто скопируйте его в каталог, входящий в `include_path`, чтобы PHP смог найти этот файл.

Ниже приведен код клиентской программы, вызывающий функцию на сервере XML-RPC, который возвращает названия штатов:

```
// это имя файла по умолчанию из пакета, хранящееся
// здесь, чтобы избежать путаницы в имени файла
require 'utils.php';

// параметры сервера
$host = 'betty.userland.com';
$port = 80;
$uri = '/RPC2';

// параметры запроса:
// передаем число из диапазона 1-50.
// в ответ получаем n-й штат в алфавитном порядке,
// где 1 это Alabama, 50 это Wyoming и т.п.
$method = 'examples.getStateName';
$args = array(32);    // передаваемые данные

// строим из этих переменных массив
$request = compact('host', 'port', 'uri', 'method', 'args');

// эта функция создает соответствующий XML-RPC запрос
```

```
$result = xu_rpc_http_concise($request);

print "I love $result!\n";
```

Обсуждение

XML-RPC – формат, созданный фирмой Userland Software, позволяет посылать запросы веб-серверу с помощью протокола HTTP. Сам запрос представляет собой XML-документ, отформатированный особым образом. В качестве клиента вы строите посылаемый XML-запрос, который должен удовлетворять спецификации XML-RPC. Затем вы посылаете его на сервер, а сервер отвечает вам XML-документом. Для извлечения результатов необходимо проанализировать полученный XML-документ. Сервер XML-RPC в разделе «Решение» возвращает название штата, поэтому программа печатает:

```
I love New York!
```

В отличие от более ранних реализаций XML-RPC, которые были запрограммированы на PHP, текущая версия встроенного расширения написана на C, и обработка с ее помощью выполняется значительно быстрее. Чтобы подключить это расширение, добавьте при конфигурировании PHP параметр `--with-xmlrpc`.

Параметры сервера указывают PHP, какому серверу посылать запрос. Переменная `$host` – это имя хоста сервера; переменная `$port` обозначает порт, через который работает веб-сервер, обычно это порт с номером 80; переменная `$uri` – это относительный путь к серверу XML-RPC, с которым нужно соединиться. Этот запрос эквивалентен запросу *http://betty.userland.com:80/RPC2*. Если порт не указан, функция по умолчанию обращается к порту 80, а в качестве URI по умолчанию берется корневой каталог веб-сервера /.

Параметрами запроса являются имя вызываемой функции и данные, которые ей передаются. Так, метод `examples.getStateName` принимает целое число из диапазона от 1 до 50 и возвращает соответствующее название американского штата в алфавитном порядке. В XML-RPC имена методов могут включать точку, а в PHP нет. Если бы это было так, то PHP-эквивалентом передачи числа 32 в качестве аргумента при вызове метода XML-RPC `examples.getStateName` был бы вызов функции `examples.getStateName()`:

```
examples.getStateName(32);
```

В XML-RPC это выглядит следующим образом:

```
<?xml version='1.0' encoding="iso-8859-1" ?>
<methodCall>
<methodName>examples.getStateName</methodName>
<params>
 <param>
  <value>
   <int>32</int>
```

```
    </value>
   </param>
  </params>
  </methodCall>
```

Параметры сервера и информация запроса заносятся в единый ассоциативный массив, который передается функции `xu_rpc_http_concise()`. Для сокращения записи выполняется вызов функции `compact()`, который идентичен следующему:

```
$request = array('host'   => $host,
                 'port'   => $port,
                 'uri'    => $uri,
                 'method' => $method,
                 'args'   => $args);
```

Функция `xu_rpc_http_concise()` выполняет вызов XML-RPC и возвращает результаты. Поскольку возвращенное значение является строкой, можно непосредственно вывести переменную `$results`. Если же наш XML-RPC-вызов возвращает множество значений, то функция `xu_rpc_http_concise()` также возвращает массив.

В массиве, передаваемом функции `xu_rpc_http_concise()`, может быть до 10 различных параметров, но только один из них обязателен – `host`. Эти параметры приведены в табл. 12.1.

Таблица 12.1. Параметры функции xu_rpc_http_concise( )

Имя	Описание
host	Имя хоста сервера
uri	URI сервера (по умолчанию /)
port	Порт сервера (по умолчанию 80)
method	Имя вызываемого метода
args	Аргументы, передаваемые методу
debug	Уровень отладки (от 0 до 2; 0 – выключена, 2 – наибольший)
timeout	Ограничение времени выполнения запроса в секундах; значение 0 – нет ограничений
user	Имя пользователя для базовой аутентификации HTTP, если необходимо
pass	Пароль для базовой аутентификации, если необходимо
secure	Для шифрования посылок используется защищенное соединение SSL; требует сборки PHP с поддержкой SSL (при указании требуется передать любое истинное значение)

См. также

Рецепт 12.7, в котором более подробно рассказывается о серверах XML-RPC; о вспомогательных функциях PHP, использующихся с рас-

ширением xmlrpc-epi, на *http://xmlrpc-epi.sourceforge.net/*; книгу «Programming Web Services with XML-RPC» Саймона С. Лаурента (Simon St. Laurent), Джо Джонсона (Joe Johnston) и Эда Думбила (Edd Dumbill) (издательство O'Reilly); дополнительную информацию о XML-RPC на *http://www.xml-rpc.com*.

12.7. Прием запросов XML-RPC

Задача

Необходимо создать сервер XML-RPC и отвечать на XML-RPC-запросы. Это позволит клиенту, поддерживающему формат XML-RPC, задавать вашему серверу вопросы, а вам посылать в ответ данные.

Решение

Используйте PHP-расширение XML-RPC. Ниже приведена PHP-версия демонстрационного XML-RPC приложения фирмы Userland, которое возвращает строку стандарта ISO 8601 с текущей датой и временем:

```
// это функция, подставляемая как "get_time()"
function return_time($method, $args) {
   return date('Ymd\THis');
}

$server = xmlrpc_server_create() or die("Can't create server");
xmlrpc_server_register_method($server, 'return_time', 'get_time')
    or die("Can't register method.");

$request = $GLOBALS['HTTP_RAW_POST_DATA'];
$options = array('output_type' => 'xml', 'version' => 'xmlrpc');

print xmlrpc_server_call_method($server, $request, NULL, $options)
    or die("Can't call method");

xmlrpc_server_destroy($server);
```

Обсуждение

Поскольку встроенное расширение XML-RPC, xmlrpc-epi, написано на C, оно обрабатывает XML-RPC-запросы быстрее и эффективнее. Чтобы подключить это расширение, добавьте к конфигурационной строке во время компиляции параметр `--with-xmlrpc`. Более подробную информацию о XML-RPC см. в рецепте 12.6.

Решение начинается с определения функции PHP, предназначенной для связи с методом XML-RPC. Имя функции — `return_time()`. Эта функция связывается с методом XML-RPC `get_time()`:

```
function return_time($method, $args) {
   return date('Ymd\THis');
}
```

Функция возвращает строку стандарта ISO 8601 с текущей датой и временем. Мы преобразуем символ T внутри вызова функции date() в escape-последовательность, т. к.к спецификация требует литерал T для разделения даты и времени. Для 21 августа 2002 года в 3:03:51 P.M. возвращаемое значение будет равно 20020821T150351.

Функция автоматически вызывается с двумя параметрами: именем метода XML-RPC, которому отвечает сервер, и массивом аргументов метода, передаваемым серверу XML-RPC-клиентом. В данном примере сервер игнорирует обе переменные.

Затем создаем сервер XML-RPC и регистрируем нужный метод – get_time():

```
$server = xmlrpc_server_create() or die("Can't create server");
xmlrpc_server_register_method($server, 'return_time', 'get_time');
```

Мы создали новый сервер и присвоили его переменной $server. Затем вызываем функцию xmlrpc_server_register_method() с тремя параметрами. Первый – это вновь созданный сервер, второй – имя регистрируемого метода, а третий – имя функции PHP, обрабатывающей запрос.

Теперь, когда все сконфигурировано, отдаем серверу команду вызвать обрабатывающий метод и вернуть результаты клиенту:

```
$request = $GLOBALS['HTTP_RAW_POST_DATA'];
$options = array('output_type' => 'xml', 'version' => 'xmlrpc');

print xmlrpc_server_call_method($server, $request, NULL, $options);
```

Данные клиентского запроса приходят методом POST. PHP конвертирует HTTP POST-данные в соответствующие переменные, но это XML-RPC-данные, и поэтому серверу необходим доступ к непроанализированным данным, хранящимся в переменной $GLOBALS['HTTP_RAW_POST_DATA']. В данном примере XML-запрос выглядит следующим образом:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<methodCall>
<methodName>get_time</methodName>
<params/></methodCall>
```

Таким образом, сервер обращается к методу get_time(), и метод не ожидает никаких параметров.

Мы также настраиваем параметры ответа, чтобы выводить результаты в виде XML и интерпретировать запрос как XML-RPC. Эти две переменные затем передаются методу xmlrpc_server_call_method() вместе с XML-RPC-сервером $server. Третьим параметром может быть любая пользовательская информация, которую нужно передать. В данном случае она отсутствует, поэтому мы передаем NULL.

Функция xmlrpc_server_call_method() декодирует переменные, вызывает соответствующую функцию для реализации метода и кодирует ответ в формате XML-RPC. Чтобы отправить ответ клиенту, нужно всего

лишь вывести возвращаемый методом xmlrpc_server_call_method() результат.

Наконец, мы завершаем работу вызовом функции:

```
xmlrpc_server_destroy($server);
```

Используя клиентскую XML-RPC-программу из рецепта 12.6, можно послать запрос и определить время так, как это показано ниже:

```
require 'utils.php';

$output = array('output_type' => 'xml', 'version' => 'xmlrpc');
$result = xu_rpc_http_concise(array(
                              'method'  => 'get_time',
                              'host'    => 'clock.example.com',
                              'port'    => 80,
                              'uri'     => '/time-xmlrpc.php',
                              'output' => $output));

print "The local time is $result.\n";
```

```
The local time is 20020821T162615.
```

Разрешается связывать несколько методов с одним XML-RPC-сервером. Можно также связать несколько методов с одной и той же функцией PHP. Например, мы создаем сервер, который отвечает двум методам: get_gmtime() и get_time(). Первый метод, get_gmtime(), подобен методу get_time(), но выдает в ответ текущее время в зоне GMT. Чтобы справиться с этой ситуацией, можно расширить метод get_time() для приема необязательного параметра – названия временной зоны, который будет использован при вычислении текущего времени.

Ниже показано, как изменить функцию return_time() для работы с обоими методами:

```
function return_time($method, $args) {
    if ('get_gmtime' == $method) {
        $tz = 'GMT';
    } elseif (!empty($args[0])) {
        $tz = $args[0];
    } else {
        // use local time zone
        $tz = '';
    }

    if ($tz) { putenv("TZ=$tz"); }
    $date = date('Ymd\THis');
    if ($tz) { putenv('TZ=EST5EDT'); }  // меняем EST5EDT на зону вашего
                                        // сервера
    return $date;
}
```

Эта функция использует и параметр $method, и параметр $args. В начале функции проверяется, использует ли запрос метод get_gmtime. Если

это так, то в качестве временной зоны выбирается зона GMT. Если нет, то устанавливается альтернативная временнбя зона, определяемая путем проверки элемента `$args[0]`. Если ни одна из проверок не возвращает значение `true`, сохраняется текущая временная зона.

Чтобы сконфигурировать сервер для работы с новым методом, добавляем только одну новую команду:

```
xmlrpc_server_register_method($server, 'return_time', 'get_gmtime');
```

Она связывает метод `get_gmtime()` с функцией `return_time()`.

Ниже приведен пример действий клиента. Первый запрос – с использованием метода `get_time()` без параметров; второй запрос вызывает метод `get_time()` с временной зоной `PST8PDT`, которая отстает от времени сервера на три часа; последний запрос использует новый метод `get_gmtime()`, который опережает временную зону сервера на четыре часа.

```
require 'utils.php';

$output = array( 'output_type' => 'xml', 'version' => 'xmlrpc');

// get_time()
$result = xu_rpc_http_concise(array(
                              'method'  => 'get_time',
                              'host'    => 'clock.example.com',
                              'port'    => 80,
                              'uri'     => '/time.php',
                              'output' => $output));

print "The local time is $result.\n";

// get_time('PST8PDT')
$result = xu_rpc_http_concise(array(
                              'method'  => 'get_time',
                              'args'    => array('PST8PDT'),
                              'host'    => 'clock.example.com',
                              'port'    => 80,
                              'uri'     => '/time.php',
                              'output' => $output));

print "The time in PST8PDT is $result.\n";

// get_gmtime()
$result = xu_rpc_http_concise(array(
                              'method'  => 'get_gmtime',
                              'host'    => 'clock.example.com',
                              'port'    => 80,
                              'uri'     => '/time.php',
                              'output' => $output));

print "The time in GMT is $result.\n";
```

```
The local time is 20020821T162615.
The time in PST8PDT is 20020821T132615.
The time in GMT is 20020821T202615.
```

См. также

Дополнительную информацию о клиентах XML-RPC в рецепте 12.6; документацию по функции `xmlrpc_server_create()` на *http://www.php.net/xmlrpc-server-create*, по функции `xmlrpc_server_register_method()` на *http://www.php.net/xmlrpc-server-register-method*, по функции `xmlrpc_server_call_method()` на *http://www.php.net/xmlrpc-server-call-method* и по функции `xmlrpc_server_destroy()` на *http://www.php.net/xmlrpc-server-destroy*; книгу «Programming Web Services with XML-RPC» Саймона С. Лаурента (Simon St. Laurent), Джо Джонсона (Joe Johnston) и Эда Думбила (Edd Dumbill) (O'Reilly); дополнительную информацию о XML-RPC на *http://www.xml-rpc.com*; XML-RPC-сервер оригинального текущего времени на *http://www.xmlrpc.com/currentTime*.

12.8. Посылка SOAP-запросов

Задача

Необходимо послать SOAP-запрос. Создание SOAP-клиентов позволяет собирать информацию с SOAP-серверов независимо от их операционных систем и связующего программного обеспечения.

Решение

Применяем SOAP-классы PEAR. Ниже показана клиентская программа, использующая SOAP-сервис GoogleSearch:

```
require 'SOAP/Client.php';

$query = 'php'; // ваши условия поиска в Google
$soap = new SOAP_Client('http://api.google.com/search/beta2');

$params = array(
            new SOAP_Value('key',         'string',  'your google key'),
            new SOAP_Value('q',           'string',  $query),
            new SOAP_Value('start',       'int',     0),
            new SOAP_Value('maxResults',  'int',     10),
            new SOAP_Value('filter',      'boolean', false),
            new SOAP_Value('restrict',    'string',  ''),
            new SOAP_Value('safeSearch',  'boolean', false),
            new SOAP_Value('lr',          'string',  'lang_en'),
            new SOAP_Value('ie',          'string',  ''),
            new SOAP_Value('oe',          'string',  ''));

$hits = $soap->call('doGoogleSearch', $params, 'urn:GoogleSearch');

foreach ($hits->resultElements as $hit) {
    printf('<a href="%s">%s</a><br />', $hit->URL, $hit->title);
}
```

Обсуждение

Простой протокол доступа к объектам (Simple Object Access Protocol, SOAP) – это способ обмена информацией, подобно XML-RPC реализованный поверх протокола HTTP. В качестве формата сообщений он также использует XML, что облегчает процесс их создания и анализа. В результате, так как SOAP не зависит от платформы и языка, он доступен на многих платформах и во многих языках, включая PHP. Чтобы сделать SOAP-запрос, нужно создать новый объект `SOAP_Client` и передать конструктору адрес страницы, на которую посылается этот запрос:

```
$soap = new SOAP_Client('http://api.google.com/search/beta2');
```

В настоящее время поддерживается два различных метода взаимодействия: HTTP и SMTP. Допускается также защищенный режим HTTP, если в вашей версии PHP встроен SSL. Для выбора используемого метода укажите его в качестве префикса вашего URL (`http`, `https` или `mailto`).

После создания объекта `SOAP_Client` используйте его метод `call()` для вызова удаленной функции:

```
$query = 'php';

$params = array(
            new SOAP_Value('key',         'string',  'your google key'),
            new SOAP_Value('q',           'string',  $query),
            new SOAP_Value('start',       'int',     0),
            new SOAP_Value('maxResults',  'int',     10),
            new SOAP_Value('filter',      'boolean', false),
            new SOAP_Value('restrict',    'string',  ''),
            new SOAP_Value('safeSearch',  'boolean', false),
            new SOAP_Value('lr',          'string',  'lang_en'),
            new SOAP_Value('ie',          'string',  ''),
            new SOAP_Value('oe',          'string',  ''));

$hits = $soap->call('doGoogleSearch', $params, 'urn:GoogleSearch');
```

Массив `$params` содержит совокупность объектов `SOAP_Value`. Объект `SOAP_Value` создается с тремя аргументами: именем, типом и значением параметра, которое передается SOAP-серверу. Они меняются от сообщения к сообщению в зависимости от SOAP-функции, доступной на сервере.

Реальные события происходят в методе `SOAP_Client::call()`, который принимает несколько аргументов. Первый – это метод, который должен выполнить сервер; в данном случае это `doGoogleSearch`. Второй аргумент – это массив параметров, передаваемых функции на SOAP-сервере. Третий аргумент, `urn:GoogleSearch`, – это пространство имен SOAP; он позволяет серверу узнать, что `doGoogleSearch` принадлежит

к пространству имен GoogleSearch. С помощью пространства имен разрешается возможный конфликт между одноименными методами.

Есть и четвертый параметр, не используемый в данном случае – soapAction. Этот параметр можно добавить, если требуется сообщить SOAP-серверу URI, представляющий цель запроса. К сожалению, определение слова «intent» трактуется от реализации к реализации по-разному. В данный момент действует соглашение о том, чтобы не использовать soapAction без дальнейшего его уточнения. SOAP-сервер PEAR не использует это поле, но другие поставщики услуг могут присвоить ему свое собственное значение.

В случае успешного выполнения функция возвращает объект, содержащий ответ сервера. При возникновении ошибки функция возвращает объект PEAR_Error. Google возвращает информацию разных типов, но в данном случае мы выполняем цикл по массиву $resultElements и извлекаем для вывода URL и название каждой найденной ссылки:

```
foreach ($hits->resultElements as $hit) {
    printf('<a href="%s">%s</a><br />', $hit->URL, $hit->title);
}
```

В результате получим:

```
<a href="http://www.php.net/"><b>PHP</b>: Hypertext Preprocessor</a>
<a href="http://www.php.net/downloads.php"><b>PHP</b>: Downloads</a>
<a href="http://phpnuke.org/"><b>PHP</b>-Nuke</a>
<a href="http://www.phpbuilder.com/">PHPBuilder.com</a>
<a href="http://php.resourceindex.com/">The <b>PHP</b> Resource Index</a>
<a href="http://www.php.com/"><b>PHP</b>.com: Home</a>
<a href="http://www.php.org/"><b>PHP</b>.org</a>
<a href="http://php.weblogs.com/"><b>PHP</b> Everywhere:</a>
<a href="http://www.php3.org/"></a>
<a href="http://gtk.php.net/"><b>PHP</b>-GTK</a>
```

Для выполнения запросов можно также использовать язык определения веб-сервисов (WSDL, Web Services Definition Language). При использовании WSDL нет необходимости перечислять ключи параметров или пространство имен SOAP:

```
require 'SOAP/Client.php';

$wsdl_url = 'http://api.google.com/GoogleSearch.wsdl';
$WSDL = new SOAP_WSDL($wsdl_url);
$soap = $WSDL->getProxy();

$hits = $soap->doGoogleSearch('your google key',$query,0,10,
                              true,'',false,'lang_en','','');
```

Этот код эквивалентен самому длинному предыдущему примеру. Объект SOAP_WSDL принимает URL для WSDL-файла GoogleSearch и автоматически загружает спецификацию с этого URL. Вместо создания и

присвоения переменной `$soap` нового объекта `SOAP_Client` вызывается метод `SOAP_WSDL::getProxy()`, создающий объект GoogleSearch.

Этот новый объект имеет методы с теми же именами, что и имена SOAP-методов GoogleSearch. Поэтому, вместо того чтобы передавать `doGoogleSearch` функции `SOAP_Client::call()` в качестве первого параметра, мы вызываем метод `$soap->doGoogleSearch()`. Значения массива `$params` становятся аргументами метода, при этом нет необходимости в инкапсуляции массива или создании объектов `SOAP_Value`. Кроме того, поскольку объект находится в WSDL-файле, то нет необходимости в указании пространства имен.

См. также

Дополнительную информацию о SOAP-серверах в рецепте 12.9; рецепт 20.10, где приводится пример SOAP-клиента в приложении PHP-GTK; о SOAP-классах PEAR на *http://pear.php.net/package-info.php?package=SOAP*; книгу «Programming Web Services with XML-RPC» Саймона С. Лаурента (Simon St. Laurent), Джо Джонсона (Joe Johnston) и Эда Думбила (Edd Dumbill) (O'Reilly); информацию о SOAP-сервисе Google на *http://www.google.com/apis/*.

12.9. Прием SOAP-запросов

Задача

Необходимо создать SOAP-сервер и отвечать на SOAP-запросы. Если ваш сервер отвечает на SOAP-запросы, то любой, кто имеет соответствующий SOAP-клиент, может посылать запросы на ваш сервер через сеть Интернет.

Решение

Используем PEAR-класс `SOAP_Server`. Приведем пример сервера, который возвращает текущую дату и время:

```
require 'SOAP/Server.php';

class pc_SOAP_return_time {
    var $method_namespace = 'urn:pc_SOAP_return_time';

    function return_time() {
        return date('Ymd\THis');
    }
}

$rt = new pc_SOAP_return_time();

$server = new SOAP_Server;
$server->addObjectMap($rt);
$server->service($HTTP_RAW_POST_DATA);
```

Обсуждение

Вот три этапа создания SOAP-сервера с помощью PEAR-класса SOAP_Server:

1. Создаем класс для работы с SOAP-методами и реализуем его.
2. Создаем экземпляр SOAP-сервера и связываем обрабатываемый объект с сервером.
3. Приказываем серверу обрабатывать запрос и отвечать SOAP-клиенту.

PEAR-класс SOAP_Server использует объекты для обработки SOAP-запросов. Обрабатывающему запрос классу необходимо указать свойство $method_namespace, которое определяет его пространство имен SOAP. В данном случае это urn:pc_SOAP_return_time. Затем методы объекта назначают имена процедурам SOAP из указанного пространства имен. Действительное имя класса, принятое в PHP, не распространяется посредством SOAP, поэтому тот факт, что имя класса и его $method_namespace идентичны, является вопросом удобства, а не необходимостью:

```
class pc_SOAP_return_time {
    var $method_namespace = 'urn:pc_SOAP_return_time';

    function return_time() {
        return date('Ymd\THis');
    }
}

$rt = new pc_SOAP_return_time();
```

После того как определен класс, создается экземпляр этого класса, чтобы связать методы с объектом SOAP-сервера. Однако перед сопоставлением процедур и методов класса необходимо создать сам объект SOAP_Server:

```
$server = new SOAP_Server;
$server->addObjectMap($rt);
$server->service($GLOBALS['HTTP_RAW_POST_DATA']);
```

После того как это сделано, вызываем функцию SOAP_Server::addObjectMap(), передавая объект в качестве параметра для того, чтобы сообщить серверу о методах, предоставляемых этим объектом. Теперь сервер готов отвечать на все SOAP-запросы из пространства имен, для которого вы определили методы.

Чтобы дать указание серверу отвечать на запрос, вызовите функцию SOAP_Server::service() и передайте SOAP-конверт. Поскольку конверт доставляется с помощью метода POST, вы передаете переменную $GLOBALS['HTTP_RAW_POST_DATA']. Таким образом, серверу предоставляется весь запрос целиком, о необходимом анализе запроса позаботится класс.

Для вызова этой процедуры с помощью PEAR SOAP-клиента используйте следующий код:

```
require 'SOAP/Client.php';
$soapclient = new SOAP_Client('http://clock.example.com/time-soap.php');
$result = $soapclient->call('return_time', array(),
                            array('namespace' => 'urn:pc_SOAP_return_time'));
print "The local time is $result.\n";
```

Его вывод будет таким:

```
The local time is 20020821T132615.
```

Чтобы расширить этот метод для чтения параметров, необходимо изменить прототип метода, включив имена параметров, а затем модифицировать клиентский запрос, добавив данные дополнительных аргументов. Следующий пример модифицирует SOAP-процедуру так, чтобы она принимала временную зону в качестве необязательного аргумента:

```
class pc_SOAP_return_time {
    var $method_namespace = 'urn:pc_SOAP_return_time';

    function return_time($tz='') {
        if ($tz) { putenv("TZ=$tz"); }
        $date = date('Ymd\THis');
        if ($tz) { putenv('TZ=EST5EDT'); } // меняем EST5EDT на зону вашего
                                           //  сервера
         return $date
    }
}
```

Второй параметр в клиентском вызове теперь принимает опцию `tz`:

```
$result = $soapclient->call('return_time', array('tz' => 'PST8PDT'),
                            array('namespace' => 'urn:pc_SOAP_return_time'));
```

С новыми настройками сервер возвращает время, отстающее от предыдущего времени на три часа:

```
20020821T202615
```

См. также

Дополнительную информацию о SOAP-клиентах в рецепте 12.8; о SOAP-классах PEAR на *http://pear.php.net/package-info.php?package=SOAP*; книгу «Programming Web Services with SOAP» (O'Reilly); оригинальное SOAP-приложение текущего времени на *http://www.soapware.org/currentTime*.

12.10. Обмен данными с помощью WDDX

Задача

Необходимо преобразовать данные в последовательный вид с помощью формата WDDX или провести обратное преобразование принятых в формате WDDX данных. Это позволит общаться со всеми, кто поддерживает формат WDDX.

Решение

Используем PHP-расширение WDDX. Переводим несколько переменных в последовательную форму с помощью функции `wddx_serialize_vars()`:

```
$a = 'string data';
$b = 123;
$c = 'rye';
$d = 'pastrami';
$array = array('c', 'd');

$wddx = wddx_serialize_vars('a', 'b', $array);
```

Также можно запустить пакет WDDX с помощью функции `wddx_packet_start()` и добавлять данные по мере их поступления с помощью функции `wddx_add_vars()`:

```
$wddx = wddx_packet_start('Some of my favorite things');

// выполняем цикл по данным
while ($array = mysql_fetch_array($r)) {
    $thing = $array['thing'];
    wddx_add_vars($wddx, 'thing');
}

$wddx = wddx_packet_end($wddx);
```

Используем функцию `wddx_deserialize()` для сериализации данных:

```
// переменная $wddx содержит пакет WDDX
$vars = wddx_deserialize($wddx);
```

Обсуждение

WDDX, расшифровываемый как формат обмена распределенной веб-информацией (Web Distributed Data eXchange), был одним из первых XML-форматов совместного использования информации, основанных на принципе независимости от языка. Придуманный компанией под эгидой ColdFusion, формат WDDX достиг большой популярности в 1999 году, но в настоящее время совсем не развивается.

Вместо WDDX многие начали использовать SOAP. Но формат WDDX имеет свое преимущество простоты. Поэтому если для вас главным является информация, которой вы обмениваетесь, то WDDX может стать

хорошим выбором. Кроме того, вследствие происхождения его легко читать и записывать в ColdFusion. Если требуется взаимодействие с приложением ColdFusion, формат WDDX будет очень полезен.

Формату WDDX требуется библиотека *expat*, доступная в Apache версии 1.3.7 и выше или на сайте *http://www.jclark.com/xml/expat.html*. Для работы с WDDX сконфигурируйте PHP с опциями `--with-xml` и `--enable-wddx`.

Пример в «Решении» выдает следующий XML-документ (отформатированный для облегчения восприятия):

```
<wddxPacket version='1.0'>
<header/>
<data>
    <struct>
        <var name='a'><string>string data</string></var>
        <var name='b'><number>123</number></var>
        <var name='c'><string>rye</string></var>
        <var name='d'><string>pastrami</string></var>
    </struct>
</data>
</wddxPacket>
```

Переменные заключены в теги `<var>`, в которых атрибуту `name` в качестве значения присвоено имя переменной. Внутри этих тегов находится ряд других, означающих тип переменных, таких как `string`, `number`, `dateTime`, `boolean`, `array`, `binary` или `recordSet`. И наконец, внутри находятся сами данные.

Кроме того, можно сериализовать одну переменную за раз с помощью функции `wddx_serialize_value`:

```
// one variable
$s = wddx_serialize_value('Serialized', 'An optional comment');
```

В результате получим следующий XML-текст:

```
<wddxPacket version='1.0'>
<header>
    <comment>An optional comment</comment>
</header>
<data>
    <string>Serialized</string>
</data>
</wddxPacket>
```

См. также

Документацию по WDDX на *http://www.php.net/wddx*; дополнительную информацию на *http://www.openwddx.org*; раздел «Sharing Data with WDDX» из книги «Programming ColdFusion» Роба Брукс-Бильсона (Rob Brooks-Bilson) (издательство O'Reilly).

12.11. Чтение RSS-рассылок

Задача

Необходимо получить RSS-рассылку и просмотреть ее содержание. Это позволит нам включить рассылку новостей с нескольких веб-сайтов в ваше приложение.

Решение

Используйте PEAR-класс XML_RSS. Приведем пример, который читает RSS-рассылку для списка почтовых адресов *php.announce*:

```
require 'XML/RSS.php';

$feed = 'http://news.php.net/group.php?group=php.announce&format=rss';

$rss =& new XML_RSS($feed);
$rss->parse();

print "<ul>\n";
foreach ($rss->getItems() as $item) {
  print '<li><a href="' . $item['link'] . '">' . $item['title'] . "</a></li>\n";
}
print "</ul>\n";
```

Обсуждение

RSS, что означает RDF Site Summary, является простым в использовании форматом заголовков или синдикации статей, написанным на XML.[1] Многие новостные веб-сайты, такие как Slashdot и Meerkat издательства O'Reilly, предоставляют RSS-рассылки, которые обновляются при каждой публикации нового материала. Популярные сетевые издания также включают RSS, а RSS-рассылка в вашем собственном онлайн-дневнике является уже стандартной функциональностью. Веб-сайт PHP также публикует RSS-рассылки для большинства серверов почтовых рассылок PHP.

Получать и анализировать RSS-рассылку просто:

```
$feed = 'http://news.php.net/group.php?group=php.announce&format=rss';

$rss =& new XML_RSS($feed);
$rss->parse();
```

Этот пример присваивает переменной `$rss` новый объект `XML_RSS` и передает значение переменной feed в RSS-рассылку новостей *php.announce*. Затем рассылка анализируется с помощью функции `XML_RSS::parse()` и сохраняется в переменной `$rss`.

---

[1] RDF означает Resource Definition Framework. RSS также означает Rich Site Summary.

Впоследствии к элементам RSS обращаются как к ассоциативному массиву, полученному с помощью функции `XML_RSS:getItems()`:

```
print "<ul>\n";

foreach ($rss->getItems() as $item) {
  print '<li><a href="' . $item['link'] . '">' . $item['title'] . "</a></li>\n";
}

print "</ul>\n";
```

Этот цикл `foreach` создает неупорядоченный список элементов ссылок на статьи так, как показано на рис. 12.1. Помимо обязательных полей `title` и `link`, элемент может иметь и необязательное поле `description`, в котором содержится краткое описание статьи.

Каждый канал также содержит вход с общей информацией о рассылке, как это показано на рис. 12.2. Для извлечения данных используем функцию `XML_RSS::getChannelInfo()`:

```
$feed = 'http://news.php.net/group.php?group=php.announce&format=rss';
$rss =& new XML_RSS($feed);

$rss->parse();

print "<ul>\n";

foreach ($rss->getChannelInfo() as $key => $value) {
    print "<li>$key: $value</li>\n";
}

print "</ul>\n";
```

См. также

Рецепт 12.4 о том, как обрабатывать RSS-рассылку и преобразовывать ее в HTML-документ; о PEAR-классе `XML_RSS` на *http://pear.php.net/package-info.php?package=XML_RSS*; дополнительную информацию о RSS на *http://groups.yahoo.com/group/rss-dev/files/specification.html*; сайт Meerkat O'Reilly Network на *http://www.oreillynet.com/meerkat/*.

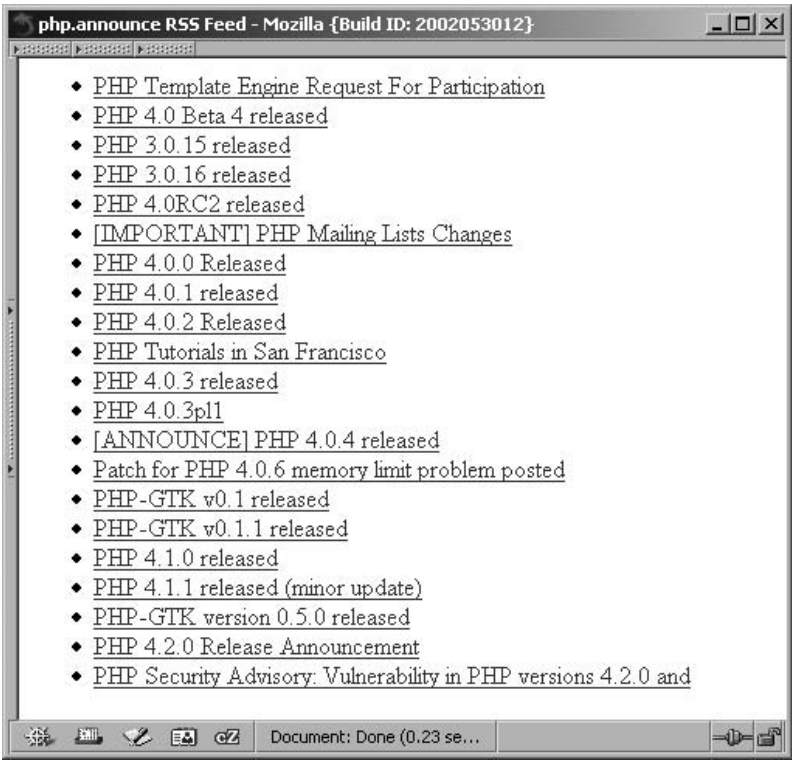

Рис. 12.1. *RSS-рассылка php.announce*

Рис. 12.2. *RSS-информационный канал php.announce*

13

Регулярные выражения

13.0. Введение

Регулярные выражения – это мощный инструмент сравнения и обработки текста. Они не так быстры, как обычные операции сравнения строк, но отличаются чрезвычайной гибкостью, позволяя создавать шаблоны сравнения для практически любой комбинации символов при помощи довольно простого, хотя и немногословного и до некоторой степени трудного для понимания синтаксиса.

В PHP функции регулярных выражений применяются для поиска текста, удовлетворяющего определенному критерию. После того как нужный текст найден, можно заменять или модифицировать как весь текст, так и его части – подстроки, соответствующие шаблону. Например, следующее регулярное выражение преобразует текстовые адреса электронной почты в гиперссылки вида `mailto:`:

```
$html = preg_replace('/[^@\s]+@([-a-z0-9]+\.)+[a-z]{2,}/i',
                     '<a href="mailto:$0">$0</a>', $text);
```

Как видите, регулярные выражения довольно удобны для преобразования простого текста в формат HTML и наоборот. К счастью, эта тема столь популярна, что в PHP уже имеется множество встроенных функций для решения таких задач. В рецепте 9.8 рассказано, как превращать сущности HTML в escape-последовательности, рецепт 11.11 посвящен удалению тегов HTML, а в рецептах 11.9 и 11.10 показано, как конвертировать ASCII в HTML и, соответственно, HTML в ASCII. Дополнительные сведения о поиске и проверке адресов электронной почты посредством регулярных выражений можно почерпнуть из рецепта 13.6.

За несколько лет функциональное назначение регулярных выражений сильно расширилось по сравнению с их базовыми возможностями и теперь включает в себя множество чрезвычайно полезных характеристик. В результате PHP в современном виде предлагает два различных набора функций для работы с регулярными выражениями. Пер-

вый набор включает традиционные (или POSIX) функции, которые начинаются с префикса ereg (от «extended regular expressions» – расширенные регулярные выражения; фактически функции ereg уже сами по себе являются расширением базового набора возможностей). Другой набор включает в себя функции Perl-диалекта регулярных выражений с префиксом preg (от «Perl-compatible regular expressions» – Perl-совместимые регулярные выражения).

Работа функций preg основана на библиотеке, имитирующей функциональность регулярных выражений языка программирования Perl. Это весьма удачное решение, поскольку Perl-диалект позволяет делать с помощью регулярных выражений массу полезных вещей, таких как непоглощающие сравнения, опережающие и ретроспективные проверки и даже рекурсивные шаблоны.

По сути, нет причин и дальше использовать функции ereg. Они обладают меньшими возможностями и более медленны, чем функции preg. Однако функции ereg существовали в PHP задолго до введения функций preg, поэтому многие программисты все еще применяют их из-за того, что приходится иметь дело с унаследованными программами, или в силу привычки. К счастью, прототипы функций двух этих наборов идентичны, поэтому можно без труда переключаться от одного набора к другому и обратно, не внося слишком большой путаницы в код. (В рецепте 13.1 мы покажем, как это делается, пропуская наиболее очевидные моменты.)

Основы регулярных выражений довольно просты для понимания. Последовательность символов заключается в шаблон. Затем строки текста сравниваются с шаблоном и определяются совпадения. Большинство символов в шаблоне представляют сами себя. Поэтому для того чтобы определить, содержит ли строка HTML тег изображения, надо сделать следующее:

```
if (preg_match('/<img /', $html)) {
    // найден открывающий тег изображения
}
```

Функция preg_match() сравнивает шаблон "<img " с содержимым переменной $html. Если она находит совпадение, то возвращает значение 1; в противном случае возвращается 0. Символы / называются разделителями *шаблона*; они определяют начало и конец шаблона.

Есть и несколько специальных символов. Причем именно их специальная природа расширяет возможности регулярных выражений по сравнению с такими функциями, как strstr() и strpos().Такие символы называют метасимволами. К наиболее часто используемым метасимволам относятся точка (.), звездочка (*), плюс (+) и знак вопроса (?). Соответственно, чтобы задать в шаблоне поиск символа, соответствующего метасимволу, необходимо поставить перед символом обратную косую черту (\).

- Точка соответствует любому символу, поэтому шаблон `/.at/` будет совпадать с `bat`, `cat` и даже с `rat`.
- Звездочка означает 0 или более повторений предшествующих объектов. (В настоящий момент единственными объектами, о которых нам известно, являются символы.)
- Действие плюса подобно действию звездочки, но означает 1 или более повторений, а не 0 или более. Поэтому `/.+at/` соответствует `brat`, `sprat` и даже `catastrophe`, но не `at`. Чтобы найти `at`, нужно заменить `+` на `*`.
- Знак вопроса означает 0 или 1 объект.

Чтобы применить `*` и `+` к объектам, состоящим более чем из одного символа, заключите эту последовательность символов в круглые скобки. Круглые скобки позволяют оперировать группами символов для более сложного поиска и сохраняют часть шаблона, заключенную в них. На выделенную последовательность можно затем ссылаться в функции `preg_replace()` для замены строки, и все совпадения, выделенные таким способом, могут быть сохранены в массиве, передаваемом в качестве третьего параметра в функции `preg_match()` и `preg_match_all()`. Функция `preg_match_all()` действует подобно функции `preg_match()`, но находит все возможные совпадения внутри строки, а не останавливается на первом. Приведем несколько примеров:

```
if (preg_match('/<title>.+<\/title>/', $html)) {
    // страница имеет название
}

if (preg_match_all('/<li>/', $html, $matches)) {
    print 'Page has ' . count($matches[0]) . " list items\n";
}

// превращаем полужирный шрифт в курсив
$italics = preg_replace('/(<\/?)b(>)/', '$1i$2', $bold);
```

Если вы хотите найти строки, состоящие из определенного набора букв, создайте символьный класс из этих букв. *Символьный класс* представляет собой последовательность символов, заключенную в квадратные скобки. Знак вставки (`^`) и знак доллара (`$`) связывают шаблон с началом и концом строки соответственно. Без их указания поиск совпадений может происходить в любом месте строки. Поэтому, для того чтобы найти все гласные в строке, надо создать символьный класс, содержащий `a`, `e`, `i`, `o` и `u`, начать шаблон с символа `^` и закончить его символом `$`:

```
preg_match('/^[aeiou]+$/', $string); // только гласные
```

Если искомое проще определить с помощью дополнения, можно так и сделать. Чтобы построить символьный класс, определяющий дополнение своего содержимого, укажите в его начале символ вставки. Если указанный снаружи символьного класса символ вставки привязывает

шаблон к началу строки, то символ вставки внутри символьного класса означает «совпадение со всем, кроме того, что перечислено в квадратных скобках»:

```
preg_match('/^[^aeiou]+$/', $string) // только не гласные
```

Обратите внимание, противоположность [aeiou] – это не только [bcdfghjklmnpqrstvwxyz]. Символьный класс [^aeiou] также соответствует гласным в верхнем регистре, таким как AEIOU, числам, например 123, URL, таким как http://www.cnpq.br/, и даже смайликам, например :).

Вертикальная черта (|), используемая также для указания канала, задает указание альтернативы, например:

```
// находим gif или jpeg
preg_match('/(gif|jpeg)/', $images);
```

Кроме рассмотренных выше, существует еще одна группа метасимволов. Они действуют схожим образом, но соответствуют не одному, а нескольким символам. Полезными метасимволами из этой группы являются: \w (соответствует любому текстовому символу, [a-zA-Z0-9_]);[1] \d (соответствует любой цифре, [0-9]); \s (соответствует любому пробельному символу) и \b (соответствует границе слова).[2] Ниже показано, как найти все числа, которые не являются частью другого слова:

```
// находим цифры, не соприкасающиеся с другими словами
preg_match_all('/\b\d+\b/', $html, $matches);
```

Это соответствует 123, 76! и 38-years-old, но не 2nd.

Приведем шаблон, представляющий собой регулярное выражение, эквивалентное функции trim():

```
// удаляем начальные или заключительные пробельные символы
$trimmed = preg_replace('/(^\s+)|(\s+$)/', '', $string);
```

Наконец, поговорим о модификаторах шаблона. Модификаторы влияют на весь шаблон, а не только на отдельный символ или группу символов. Модификаторы шаблона располагаются после закрывающего

---

[1] В оригинале – «any word character», что можно перевести как «любой текстовый символ». Данная нотация заимствована из языка Perl. Фактически множество символов \w трактуется в нем как множество символов, допустимых в идентификаторах (именах переменных). Это изначально соответствует символам английского алфавита в верхнем и нижнем регистрах, цифрам и символу подчеркивания. Это множество принято также называть «символами слова». – *Примеч. науч. ред.*

[2] Несмотря на название, в действительности символ \b определяется как позиция между символами, по одну сторону от которой расположены символы множества \w, а по другую – символы дополнения этого множества. Например, в результате вычисления функции preg_match_all("/(\b\w+\b)/", "38-years-old", $match) массив $match получит значение ("38", "years", "old"). – *Примеч. науч. ред.*

разделителя шаблона. Например, буква `i` делает регулярное выражение в шаблоне не чувствительным к регистру:

```
// точное совпадение только с тегами изображения
// в нижнем регистре (допускается в XHTML)
if (preg_match('/<img[^>]+>/', $html)) {
   ...
}

// соответствует тегам изображения и в верхнем, и в нижнем регистре
if (preg_match('/<img[^>]+>/i', $html)) {
   ...
}
```

Мы затронули лишь небольшое подмножество регулярных выражений. Некоторые дополнительные аспекты мы рассмотрим в следующих рецептах, а на веб-сайте PHP, по адресу *http://www.php.net/regex* можно найти очень полезную информацию о регулярных выражениях POSIX, тогда как регулярные выражения, поддерживающие диалект Perl, рассмотрены по адресу *http://www.php.net/pcre*. На этой последней странице приведены также ссылки на очень подробные и информативные разделы «Pattern Modifiers» (Модификаторы шаблонов) и «Pattern Syntax» (Синтаксис шаблонов).

Как лучшие книги по этой теме назовем «Mastering Regular Expressions»[1] Джеффри Фридла (Jeffrey Friedl) и «Programming Perl»[2] Ларри Уолла (Larry Wall), Тома Кристиансена (Tom Christiansen) и Джона Орванта (Jon Orwant), опубликованные издательством O'Reilly. (Регулярные выражения, совместимые с Perl, основаны на регулярных выражениях языка Perl, поэтому мы сочли целесообразным упомянуть книгу по Perl.)

13.1. Переход от ereg к preg

Задача

Необходимо отказаться от применения функций `ereg` в пользу функций `preg`.

Решение

Сначала добавим в шаблон ограничители:

```
preg_match('/pattern/', 'string')
```

[1] Фридл. Д. «Регулярные выражения», 2-е издание. – Пер. с англ. – СПб: Питер, 2002.

[2] Уолл Л., Кристиансен Т., Орвант Д. «Программирование на Perl». – Пер. с англ. – СПб: Символ-Плюс, 2002.

Для не чувствительного к регистру сравнения с помощью функции `eregi()` укажите модификатор `/i`:

```
preg_match('/pattern/i', 'string');
```

Если целые числа выступают вместо строк в качестве шаблонов или замещающих значений, преобразуйте число в шестнадцатеричную форму и укажите его с помощью escape-последовательности:

```
$hex = dechex($number);
preg_match("/\x$hex/", 'string');
```

Обсуждение

Есть несколько существенных различий между `ereg` и `preg`. Во-первых, в случае применения функций `preg` шаблон представляет собой не просто строку `pattern`, а выглядит как `/pattern/`,[1] поскольку в нем должны быть указаны разделители, как в языке Perl. Следовательно:

```
ereg('pattern', 'string');
```

превращается в:

```
preg_match('/pattern/', 'string');
```

При выборе разделителей шаблона нельзя размещать символы-разделители внутри самого шаблона регулярного выражения, иначе вы слишком рано закроете шаблон. Если обойти это препятствие не удается, то любое вхождение символов-разделителей в шаблон надо экранировать с помощью символа обратной косой черты. Вместо того чтобы делать это вручную, вызовите функцию `addcslashes()`.

Например, если в качестве разделителя выступает символ `/`:

```
$ereg_pattern = '<b>.+</b>';
$preg_pattern = addcslashes($ereg_pattern, '/');
```

Значение переменной `$preg_pattern` теперь равно `<b>.+<\/b>`.

Среди функций `preg` нет аналогичного набора функций, не чувствительных к регистру. Все они могут содержать не чувствительный к регистру модификатор. Для перехода к другому набору функций замените:

```
eregi('pattern', 'string');
```

на:

```
preg_match('/pattern/i', 'string');
```

Изменение осуществляется добавлением символа `i` после завершающего разделителя.

---

[1] Или `{}`, `<>`, `||`, `##` или любая другая пара разделителей, которую вы предпочтете. PHP поддерживает их все.

Наконец, еще одно, последнее, не совсем явное различие. Если в функции `ereg_replace()` в качестве шаблона или замещающего значения выступает число (а не строка), то предполагается, что этим вы указываете ASCII-код символа. Поэтому, а также потому, что 9 – это ASCII-представление символа табуляции (т. е. `\t`), следующий код вставит символ табуляции в начало каждой строки:

```
$tab = 9;
$replaced = ereg_replace('^', $tab, $string);
```

А таким способом будет выполнено преобразование символов перевода строки:

```
$converted = ereg_replace(10, 12, $text);
```

Чтобы предотвратить такое поведение функции, в функциях `ereg` запись должна быть такой:

```
$tab = '9';
```

С другой стороны, функция `preg_replace()` рассматривает число 9 как число 9, а не как заместитель символа табуляции. Чтобы конвертировать коды символов для использования в функции `preg_replace()`, преобразуйте их в шестнадцатеричную форму и добавьте к ним префикс `\x`. Например, `9` становится `\x9` или `\x09`, а `12` превращается в `\ x0c`. В качестве альтернативы можно использовать `\t`, `\r` и `\n` для обозначения табуляции, возврата каретки и перевода строки соответственно.

См. также

Документацию по функции `ereg()` на *http://www.php.net/ereg*, по функции `preg_match()` на *http://www.php.net/preg-match* и по функции `addcslashes()` на *http://www.php.net/addcslashes*.

13.2. Поиск слов

Задача

Необходимо выделить все слова в строке.

Решение

Ключ к решению этой задачи в том, чтобы аккуратно определить, что именно мы понимаем под словом. Сформулировав это определение, используйте специальные символьные типы для создания регулярного выражения:

```
/\S+/          // все, что не является пробельным символом
/[A-Z'-]+/i    // все буквы в нижнем и верхнем регистре, апострофы и дефисы
```

Обсуждение

Простой вопрос «что же такое слово?» неожиданно оказывается довольно сложным. Хотя регулярные выражения, совместимые с Perl, имеют встроенный символьный тип для слова, определяемый символами `\w`, важно точно понимать, как определяет слово PHP. В противном случае полученный результат может оказаться для вас неожиданным.

Обычно, поскольку это непосредственно следует из определения слова в Perl, символ `\w` подразумевает все буквы, цифры и символ подчеркивания; это значит, что `a_z` словом является, а адрес электронной почты, `php@example.com`, — нет.

В этом рецепте мы рассматриваем только английские слова, но в других языках используется отличный алфавит. Поскольку регулярные выражения диалекта Perl учитывают текущие указания национальной настройки, то изменение в национальной конфигурации может сменить определение буквы, изменяя, таким образом, и определение слова.

Для борьбы с этим можно явным образом перечислить символы, принадлежащие словам, внутри символьного класса. Нестандартные символы можно добавить при помощи кодировки `\ddd`, где `ddd` представляет восьмеричный код символа.

См. также

Информацию о национальных установках в рецепте 16.2.

13.3. Нахождение n-го совпадения

Задача

Необходимо найти не первое вхождение слова, а *n*-ое.

Решение

Вызовите функцию `preg_match_all()`, чтобы занести все совпадения в массив; затем извлеките то совпадение, которое вас интересует:

```
preg_match_all ("/$pattern/$modifiers", $string, $matches)

foreach($matches[1] as $match) {
    print "$match\n";
}
```

Обсуждение

В отличие от Perl, Perl-совместимые регулярные выражения в PHP не поддерживают модификатор `/g`, позволяющий выполнить цикл поис-

ка всех совпадений в строке за один раз. Вместо функции preg_match() здесь необходимо использовать функцию preg_match_all().

Функция preg_match_all() возвращает двумерный массив. Первый элемент содержит массив совпадений с полным шаблоном. Второй элемент также содержит массив совпадений, но на более мелком уровне подсовпадений, заключенных в круглые скобки внутри каждого полного совпадения. Так, чтобы получить третью пару к слову potato, надо извлечь третий элемент из второго элемента массива $matches:

```
$potatoes = 'one potato two potato three potato four';
preg_match_all("/(\w+)\s+potato\b/", $potatoes, $matches);
print $matches[1][2];
three
```

Вместо того чтобы возвращать массив, содержащий отдельно полные совпадения и подсовпадения, функция preg_match_all() может возвратить массив, состоящий из отдельных массивов для каждого совпадения, включающих все его внутренние подсовпадения. Чтобы указать такой способ поведения функции, передайте в качестве четвертого аргумента параметр PREG_SET_ORDER. Теперь three содержится не в элементе $matches[1][2], как раньше, а в элементе $matches[2][1].

Чтобы определить общее количество совпадений, проверьте значение, возвращенное функцией preg_match_all():

```
print preg_match_all("/(\w+)\s+potato\b/", $potatoes, $matches);
3
```

Обратите внимание, что совпадений только три, а не четыре, поскольку после слова four в строке нет завершающего слова potato.

См. также

Документацию по функции preg_match_all() на *http://www.php.net/preg-match-all*.

13.4. Выбор между поглощающим и непоглощающим сравнением

Задача

Необходим шаблон для выделения наименьшей из возможных строк, а не наибольшей.

Решение

Добавьте символ ? после квантификатора, чтобы модифицировать конкретную часть шаблона:

```
// найти все части, выделенные полужирным шрифтом
preg_match_all('#<b>.+?</b>#', $html, $matches);
```

Или укажите в конце шаблона модификатор U, чтобы превратить все поглощающие квантификаторы в непоглощающие:

```
// найти все части, выделенные полужирным шрифтом
   preg_match_all('#<b>.+</b>#U', $html, $matches);
```

Обсуждение

По умолчанию все регулярные выражения в PHP представляют собой так называемые *поглощающие* регулярные выражения. Это означает, что квантификатор всегда пытается найти совпадение с максимально возможным количеством символов.

Например, возьмем шаблон p.*, означающий p, а затем 0 или более символов, и применим его к строке php. Поглощающее регулярное выражение найдет одно совпадение, поскольку после захвата начального символа p оно продолжит работу и выделит также символы hp. С другой стороны, непоглощающее регулярное выражение найдет пару совпадений. Как и раньше, оно выделит символ p и символ h, но затем, вместо того чтобы продлить свое действие, оно прервется и оставит завершающий символ p непоглощенным. Следующая, вторая проверка выбирает последнюю букву.

Следующий фрагмент программы показывает, как поглощающее сравнение приводит только к одному успешному результату, а непоглощающее сравнение находит два совпадения:[1]

```
print preg_match_all('/p.*/', "php", $match);  // поглощающее
print preg_match_all('/p.*?/', "php", $match); // непоглощающее
print preg_match_all('/p.*/U', "php", $match); // непоглощающее
1
2
2
```

Поглощающее сравнение известно также как максимальное *сравнение*, а непоглощающее сравнение по аналогии можно назвать *минимальным сравнением*, поскольку эти варианты поиска обнаруживают или максимально возможное количество подходящих символов, или минимально возможное количество.

Изначально все регулярные выражения были строго поглощающими. Поэтому нельзя использовать этот синтаксис с функциями ereg() или ereg_replace(). Поглощающее сравнение не поддерживается более старыми версиями механизма реализации функций регулярных выражений; вместо этого нужно использовать функции диалекта Perl.

---

[1] В оригинале во всех трех примерах в записи функции preg_match_all() опущен третий обязательный параметр. В русском переводе код изменен, иначе интерпретатор выдаст предупреждение и не выполнит код примера. – *Примеч. науч. ред.*

Непоглощающее сравнение часто оказывается полезным при попытке выполнения упрощенного анализа HTML-документов. Предположим, что требуется найти весь текст между тегами полужирного шрифта. В случае применения поглощающего сравнения это будет выглядеть так:

```
$html = '<b>I am bold.</b> <i>I am italic.</i> <b>I am also bold.</b>';
preg_match_all('#<b>(.+)</b>#', $html, $bolds);
print_r($bolds[1]);
Array
(
    [0] => I am bold.</b> <i>I am italic.</i> <b>I am also bold.

)
```

Поскольку здесь есть второй набор тегов полужирного шрифта, шаблон продлит свое действие за пределы первого тега `</b>`, что сделает невозможным корректный анализ HTML. В случае же минимальной проверки оба набора тегов будут найдены по шаблону и замкнуты:

```
$html = '<b>I am bold.</b> <i>I am italic.</i> <b>I am also bold.</b>';
preg_match_all('#<b>(.+?)</b>#', $html, $bolds);
print_r($bolds[1]);
Array
(
    [0] => I am bold.
    [1] => I am also bold.
)
```

Конечно, из этого может ничего не получиться, если разметка не на 100% формально корректна и в ней присутствуют случайные теги полужирного шрифта.[1] Если единственной вашей целью является удаление всех (или некоторых) тегов HTML из блока текста, то лучше вообще обойтись без регулярных выражений. Вместо этого обратитесь к функции `strip_tags()`, — она работает быстрее и корректнее. Дополнительную информацию можно найти в рецепте 11.11.

Заметим в заключение, что хотя идея непоглощающего сравнения пришла из языка Perl, сам модификатор `-U` не совместим с Perl и уникален именно для реализации Perl-диалекта регулярных выражений в PHP. Он инвертирует все квантификаторы, превращая их из поглощающих в непоглощающие, а также меняет направление их действия на обратное. Поэтому, чтобы применить поглощающий квантификатор внутри шаблона, находящегося в области действия завершающего модификатора `/U`, просто добавьте в конце квантификатора символ `?` точно так же, как это обычно делается для превращения квантификатора из поглощающего в непоглощающий.

---

[1] Даже имея формально корректный документ HTML, можно столкнуться с трудностями. Например, если теги полужирного шрифта находятся внутри комментария. Корректный HTML-анализатор проигнорирует эти разделы, но наш шаблон этого не сделает.

См. также

Дополнительную информацию о выделении текста внутри тегов HTML в рецепте 13.8; рецепт 11.11, где более подробно описывается удаление тегов HTML; документацию по функции `preg_match_all()` на *http://www.php.net/preg-match-all*.

13.5. Проверка правильности адресов электронной почты

Задача

Необходимо проверить, является ли адрес электронной почты правильным.

Решение

Это очень популярный вопрос, и каждый отвечает на него по-своему, в зависимости от собственного определения правильности. Если правильным считать имя почтового ящика, принадлежащего законному пользователю узла с реально существующим именем, то, честно говоря, вы не сможете ответить на этот вопрос корректно, можете даже не беспокоиться. Однако иногда регулярные выражения помогают избавиться от некоторых простых и очевидных попыток мошенничества. Приведем наш любимый шаблон, упомянутый выше и не требующий подробных пояснений:

```
/^[^@\s]+@([-a-z0-9]+\.)+[a-z]{2,}$/i
```

Если доступно расширение IMAP, можно также использовать функцию `imap_rfc822_parse_adrlist()`:

```
$parsed = imap_rfc822_parse_adrlist($email_address, $default_host)
if ('INVALID_ADDRESS' == $parsed['mailbox']) {
    // неправильный адрес
}
```

Ирония заключается в том, что эта функция так уступчива к RFC, что может выдать совсем не тот результат, который вы от нее ожидаете.

Обсуждение

Шаблон в данном разделе «Решение» принимает любой адрес электронной почты, в котором имя ящика представлено произвольной последовательностью символов, отличных от `@` и пробельных символов. После символа `@` должно следовать по крайней мере одно имя домена, состоящее из букв `a-z`, цифр `0-9` и дефиса, и любое количество имен поддоменов, разделенных точками. Наконец, все должно завершаться двухбуквенным кодом страны или иным доменом верхнего уровня, например `.com` или `.edu`.

Шаблон, приведенный в разделе «Решение», удобен, поскольку он будет работать, даже если ICANN создаст новые домены верхнего уровня. Тем не менее он способен пропустить некоторые ошибочные имена. Следующий, более строгий шаблон явным образом перечисляет домены верхнего уровня, не относящиеся в настоящее время к доменам стран:

```
/
    ^                   # начальная привязка
    [^@\s]+             # именем являются все символы, за исключением @
                          и пробельных символов
    @                   # символ @ отделяет имя от домена
    (
        [-a-z0-9]+      # поддомены - это буквы, цифры и дефис,
        \.              # разделенные точкой,
    )+                  # и мы можем указать один или больше
    (
        [a-z]{2}        # домены верхнего уровня
                        # могут быть двухбуквенным кодом страны
        |com|net        # или одной
        |edu|org        # из многих
        |gov|mil        # возможных
        |int|biz        # трехбуквенных
        |pro            # комбинаций
        |info|arpa      # или даже
        |aero|coop      # нескольких
        |name           # четырехбуквенных комбинаций
        |museum         # плюс одна комбинация длиной в шесть букв!
    )
    $                   # привязка в конце
/ix                     # и все это безотносительно к регистру
```

Оба шаблона умышленно либеральны к принимаемой ими информации, поскольку мы предполагаем, что вам нужно лишь, чтобы кто-нибудь случайно не потерял домен верхнего уровня или не впечатал какую-нибудь импровизацию, вроде «не указывается». Например, домен `"-.com"` не существует, но адрес `"foo@-.com"` пройдет без звука. (Нетрудно было бы модифицировать шаблон, чтобы исправить эту ситуацию, но мы оставим это читателям в качестве упражнения.) С другой стороны, допустимым является адрес `"Tim O'Reilly@oreilly.com"`, но наш шаблон его не пропустит. Однако пробелы в адресах электронной почты встречаются не часто, а т. к. пробел почти всегда является ошибкой, мы отмечаем этот адрес как ошибочный.

Каноническое определение формально корректного адреса описано в RFC 822, однако написание программы для обработки всех возможных случаев – задача не из приятных. Вот лишь один пример: придется принять во внимание, что пользователям разрешается вставлять в адреса комментарии! Комментарии заключаются в круглые скобки, поэтому допустимо написать:

```
Tim (is the man @ computer books) @ oreilly.com
```

Это эквивалентно адресу "tim@oreilly.com". (Увы, наш шаблон опять забракует этот адрес.)

Есть и альтернатива – расширение IMAP включает в себя удовлетворяющий RFC 822 анализатор адреса. Этот анализатор успешно справляется с пробельными символами и другими причудами, но допускает очевидные ошибки, поскольку предполагает, что адреса без указания имени хоста являются локальными:

```
$email = 'stephen(his account)@ example(his host)';
$parsed = imap_rfc822_parse_adrlist($email,'');
print_r($parsed);
Array
(
    [0] => stdClass Object
        (
            [mailbox] => stephen
            [host] => example
            [personal] => his host
        )

)
```

Собрав снова почтовый адрес и хост, вы получите "stephen@example", что, возможно, представляет собой не то, что ожидалось. Пустая строка, которую необходимо передать в качестве второго аргумента, лишает вас возможности проверять действительность имен хостов.

Некоторые стараются осуществить дополнительную фоновую проверку, посылая DNS-серверу запрос, зарегистрирован ли данный адрес. Большого смысла в этом нет, т. к. технически это не всегда выполнимо, и вы можете не допустить на свой сайт ни в чем не повинных и совершенно законных пользователей, действуя таким образом. (Кроме того, маловероятно, чтобы почтовый администратор выстроил собственную систему обработки почты с целью обойти схему проверки достоверности почты лишь одного сайта.)

Проверяя достоверность адресов электронной почты, надо принимать во внимание, что для пользователя не составляет большого труда ввести абсолютно законный и работающий адрес, принадлежащий кому-то другому. Например, один из авторов имел нехорошую привычку вводить "billg@microsoft.com", заходя на веб-сайты Microsoft (а вдруг Билл Гейтс не знает об этой новой версии Internet Explorer?).

Если больше всего вы беспокоитесь из-за возможных опечаток, заставьте пользователей вводить адрес дважды и сравнивайте результаты ввода. Если они совпадают, то, возможно, являются верными. Кроме того, отсекайте часто используемые фиктивные адреса, такие как "president@whitehouse.gov" и упомянутый выше адрес "billg@microsoft.com". (Не стоит волноваться о том, что президент Соединенных Штатов или Билл Гейтс не получат писем, зарегистрировавшись на вашем сайте.)

Однако если вы хотите проверить, действительно ли пользователи имеют доступ к почтовому ящику, адрес которого они указали, то одним из способов убедиться в этом является отправка на их адрес сообщения с просьбой прислать ответ или посетить страницу вашего сайта и ввести специальный код, напечатанный в теле сообщения, для подтверждения регистрации. Если вы предпочтете вариант со специальным кодом, то рекомендуем не генерировать абсолютно случайную строку символов, похожую на `HSD5nbAD18`. Это напоминает мусор, и правильно повторить его ввод довольно трудно. Вместо этого используйте список слов; создавайте кодовые слова, такие как `television4coatrack`. Несмотря на то что иногда вполне удается воссоздать наборы, используемые при составлении этих комбинаций, вы сможете резко сократить количество ошибок при вводе и собственные затраты на поддержку пользователей.

См. также

Рецепт 8.5 о генерировании хороших паролей; рецепт 8.26 о программе запрещения учетных записей веб-сайта; документацию по функции `imap_rfc822_parse_adrlist()` на *http://www.php.net/imap-rfc822-parse-adrlist*.

13.6. Поиск в файле всех строк, соответствующих шаблону

Задача

Необходимо найти в файле все строки, соответствующие шаблону.

Решение

Прочитайте файл в массив и примените функцию `preg_grep()`.

Обсуждение

Есть два способа сделать это. Рассмотрим тот, что побыстрее:

```
$pattern = "/\bo'reilly\b/i"; // только книги издательства O'Reilly
$ora_books = preg_grep($pattern, file('/path/to/your/file.txt'));
```

Используйте команду `file()` для автоматической загрузки каждой строки файла в отдельный элемент массива и функцию `preg_grep()` для отфильтровывания неподходящих строк.

Приведем более эффективный способ:

```
$fh = fopen('/path/to/your/file.txt', 'r') or die($php_errormsg);
while (!feof($fh)) {
    $line = fgets($fh, 4096);
    if (preg_match($pattern, $line)) { $ora_books[] = $line; }
```

```
}
fclose($fh);
```

Так как первый способ читает все за один раз, то он в три раза быстрее второго способа, который анализирует файл строка за строкой, но расходует меньше памяти. Однако его оборотной стороной будет то, что поскольку регулярные выражения обрабатывают лишь одну строку за раз, второй способ не будет находить совпадения, занимающие несколько строк.

См. также

Рецепт 18.5 о чтении файла в строку; документацию по функции `preg_grep()` на *http://www.php.net/preg-grep*.

13.7. Сборка текста, заключенного в теги HTML

Задача

Необходимо вычленить текст, находящийся внутри тегов HTML. Например, требуется найти все заголовки в HTML-документе.

Решение

Прочитайте HTML-файл в строку и используйте в шаблоне непоглощающее сравнение:

```
$html = join('',file($file));
preg_match('#<h([1-6])>(.+?)</h\1>#is', $html, $matches);
```

В этом примере элемент `$matches[2]` содержит массив найденных заголовков.

Обсуждение

Посредством простого регулярного выражения трудно правильно проанализировать HTML-документ. В этом состоит одно из преимуществ XHTML; с его помощью значительно легче проверить действительность (validity) документа и провести анализ.

Так, шаблон в разделе «Решение» достаточно изощрен, чтобы найти только соответствующие заголовки, поэтому `<h1>Dr. Strangelove<h1>` удовлетворяет шаблону, поскольку он заключен в теги `<h1>`, а `<h2>How I Learned to Stop Worrying and Love the Bomb</h3>` не удовлетворяет, так как в качестве открывающего тега использован `<h2>`, а в качестве закрывающего – другой тег (`</h3>`).

Эта технология работает также при нахождении текста, заключенного в теги полужирного шрифта и в теги курсива:

```
$html = join('',file($file));
preg_match('#<([bi])>(.+?)</\1>#is', $html, $matches);
```

Однако это не помогает в случае вложенных заголовков. Применение данного регулярного выражения к следующей строке:

```
<b>Dr. Strangelove or: <i>How I Learned to Stop Worrying
                       and Love the Bomb</i></b>
```

не выделяет текст внутри тегов <i> как самостоятельный элемент.

До сих пор это не было проблемой, поскольку заголовки представляют собой элементы уровня блока и не могут быть вложенными. Однако для линейных элементов, таких как выделение полужирным шрифтом или курсивом, вложение тегов вполне допустимо.

Найденный текст можно обработать, выполняя цикл по массиву совпадений. Например, следующий фрагмент кода анализирует документ на предмет заголовков и красиво их печатает с отступами, соответствующими уровню заголовков:

```
$html = join('',file($file));
preg_match('#<h([1-6])>(.+?)</h\1>#is', $html, $matches);

for ($i = 0, $j = count($matches[0]); $i < $j; $i++) {
  print str_repeat(' ', 2 * ($matches[1][$i] - 1)) . $matches[2][$i] . "\n";
}
```

В случае такого применения этого рецепта к HTML-тексту вида:

```
$html =<<<_END_
<h1>PHP Cookbook</h1>

Other Chapters
<h2>Regular Expressions</h2>

Other Recipes
<h3>Capturing Text Inside of HTML Tags</h3>

<h4>Problem</h4>
<h4>Solution</h4>
<h4>Discussion</h4>
<h4>See Also</h4>

_END_;

preg_match_all('#<h([1-6])>(.+?)</h\1>#is', $html, $matches);

for ($i = 0, $j = count($matches[0]); $i < $j; $i++) {
  print str_repeat(' ', 2 * ($matches[1][$i] - 1)) . $matches[2][$i] . "\n";
}
```

Получаем:

```
PHP Cookbook
  Regular Expressions
    Capturing Text Inside of HTML Tags
      Problem
      Solution
      Discussion
      See Also
```

Выделяя уровень заголовка и сам текст заголовка по отдельности, можно получить прямой доступ к уровню и трактовать его как число при определении размера отступа. Чтобы избежать отступа в два пробела для всех строк, вычтите из уровня 1.

См. также

Рецепт 11.7 о разметке веб-страниц и рецепт 11.8 об извлечении ссылок из HTML-файла; документацию по функции `preg_match()` на *http://www.php.net/preg-match* и по функции `str_repeat()` на *http://www.php.net/str-repeat*.

13.8. Экранирование специальных символов внутри регулярного выражения

Задача

Необходимо трактовать такие символы, как `*` или `+`, внутри регулярного выражения не как метасимволы, а как литералы. Это полезно, если пользователи вводят в строке поиска то, что впоследствии будет использовано в регулярном выражении.

Решение

Метасимволы Perl-совместимого регулярного выражения экранируются при помощи функции `preg_quote()`:

```
$pattern = preg_quote('The Education of H*Y*M*A*N K*A*P*L*A*N').':(\d+)';
if (preg_match("/$pattern/",$book_rank,$matches)) {
    print "Leo Rosten's book ranked: ".$matches[1];
}
```

Функция `quotemeta()` позволяет экранировать метасимволы POSIX:

```
$pattern = quotemeta('M*A*S*H').':[0-9]+';
if (ereg($pattern,$tv_show_rank,$matches)) {
    print 'Radar, Hot Lips, and the gang ranked: '.$matches[1];
}
```

Обсуждение

Вот символы, которые функция `preg_quote()` превращает в escape-последовательности:

```
. \ + * ? ^ $ [ ] ( ) { } < > = ! | :
```

А эти символы будут экранированы функцией `quotemeta()`:

```
. \ + * ? ^ $ [ ] ( )
```

Данные функции преобразуют метасимволы в escape-последовательности посредством добавления символа обратной косой черты.

Функция quotemeta() не находит в точности все метасимволы POSIX. Символы {, } и | также являются допустимыми метасимволами, но они не конвертируются этой функцией. Это еще одна хорошая причина применять функцию preg_match() вместо функции ereg().

Можно также передать функции preg_quote() в качестве второго аргумента дополнительный символ, чтобы преобразовать в escape-последовательность и его. Довольно полезно передать в качестве такого аргумента разделитель шаблона (обычно /), чтобы также превратить его в escape-последовательность. Это особенно важно, если в состав шаблона регулярного выражения требуется включить пользовательский ввод. Следующий фрагмент программы принимает из веб-формы строку $_REQUEST['search_term'] и ищет в строке $s слова, начинающиеся с $_REQUEST['search_term']:

```
$search_term = preg_quote($_REQUEST['search_term'],'/');
if (preg_match("/\b$search_term/i",$s)) {
   print 'match!';
}
```

Применение функции preg_quote() гарантирует правильность интерпретации регулярного выражения, если, например, поклонник Магнума П. И. (Magnum, P.I.) введет t.c. в качестве условия поиска. Без функции preg_quote() это будет соответствовать tic, tucker и другим словам, в которых первая буква t, а третья буква c. Передача разделителя шаблона функции preg_quote() также гарантирует, что пользовательский ввод с обратной косой чертой в нем, например CP/M, также будет обработан корректно.

См. также

Документацию по функции preg_quote() на *http://www.php.net/preg-quote* и по функции quotemeta() на *http://www.php.net/quotemeta*.

13.9. Чтение записей с шаблоном-разделителем

Задача

Необходимо прочитать записи из файла, в котором каждая запись отделена шаблоном, который можно сопоставить с регулярным выражением.

Решение

Прочитайте весь файл в строку, а затем выделите из нее регулярное выражение:

```
$filename = '/path/to/your/file.txt';
$fh = fopen($filename, 'r') or die($php_errormsg);
$contents = fread($fh, filesize($filename));
```

```
fclose($fh);

$records = preg_split('/[0-9]+\) /', $contents);
```

Обсуждение

Этот пример разделяет нумерованный список на части и помещает отдельные элементы списка в элементы массива. Поэтому если есть такой список:

```
1) Gödel
2) Escher
3) Bach
```

то в результате работы примера получается массив из четырех элементов с пустым первым элементом. Дело в том, что функция `preg_split()` предполагает наличие разделителей между элементами, но в данном случае элементы предваряются числами:

```
Array
(
    [0] =>
    [1] => Gödel
    [2] => Escher
    [3] => Bach
)
```

Это можно рассматривать как дополнительную функциональность, а не как ошибку, поскольку теперь *n*-й элемент массива содержит *n*-й элемент списка. Но чтобы сделать массив более компактным, можно удалить первый элемент:

```
$records = preg_split('/[0-9]+\) /', $contents);
array_shift($records);
```

Можно также выполнить другое преобразование, а именно удалить из всех элементов символ новой строки и заменить его пустой строкой:

```
$records = preg_split('/[0-9]+\) /', str_replace("\n",'',$contents));
array_shift($records);
```

PHP не позволяет изменить при вводе записей разделитель на что-нибудь другое, кроме символа новой строки, поэтому данный прием также полезен при поиске отдельных записей, разделенных переводами строки. Однако если вы просто разделяете записи внутри одной строки, то следует предпочесть более эффективную функцию `explode()`, а не `preg_split()`.

См. также

Рецепт 18.5 о чтении из файла; рецепт 1.11 об анализе CSV-файлов.

14

Шифрование и безопасность

14.0. Введение

Если бы мир был совершенным, то и шифровать ничего не пришлось бы. Любопытство людей не выходило бы за рамки их собственных данных, а номера кредитных карт, плавающие в море Интернета, не привлекали бы особого внимания. Однако наш мир не совершенен со всех точек зрения, поэтому без шифрования нам не обойтись.

Любое шифрование перестраивает данные. Данные, перестроенные определенным образом, не могут быть восстановлены без огромных дополнительных затрат. Такой способ называется односторонним *шифрованием*. Другие способы шифрования работают в обоих направлениях: сначала информация кодируется, затем она может быть декодирована.

PHP предоставляет инструменты для шифрования и обеспечения безопасности данных. Некоторые инструменты, такие как функции `crypt()` и `md5()`, входят в базовый набор функций, а некоторые относятся к расширениям, подлежащим явному добавлению во время компиляции PHP (например, *mcrypt*, *mhash* и cURL).

Функция `crypt()` выполняет одностороннее DES-шифрование, используя первые восемь символов простого текста для вычисления зашифрованного текста. Вы передаете ей обычный текст для шифрования (и так называемую «соль», или базис – параметр, усиливающий шифрование), а она возвращает зашифрованный текст. Если второй параметр не указан, PHP автоматически сгенерирует случайную пару символов для базиса шифрования:

```
print crypt('shrimp','34');
34b/4qaoXmcoY
```

Если константа `CRYPT_MD5` установлена в 1, то функция `crypt()` будет выполнять MD5-шифрование. Чтобы дать команду PHP использовать MD5-шифрование, задайте базис, начинающийся со строки `$1$`:

```
print crypt('shrimp','$1$seasalt!');
$1$seasalt!$C8bRD475BC3T4EvjjmR9I.
```

Функция `crypt()` обсуждается в рецепте 14.4. Наиболее широко она применяется для шифрования паролей.

Библиотека *mcrypt* включает больший набор возможностей, предлагая различные алгоритмы и режимы шифрования. Библиотека *mcrypt* поддерживает различные типы шифрования, поэтому она особенно полезна при обмене шифрованными данными с другими системами или с программами, написанными не на PHP. Библиотека *mcrypt* подробно рассматривается в рецепте 14.7.

PHP предоставляет инструмент для защиты данных с помощью надежного шифрования, но шифрование – это только часть большой и довольно сложной картины безопасности в целом. Зашифрованные данные могут быть вскрыты при помощи ключа, поэтому очень важно этот ключ защитить. Если ключи шифрования доступны неавторизованным пользователям (например, если они хранятся в файле, доступном через ваш веб-сервер, или в файле, доступном другим пользователям через общее окружение хостинга), ваши данные в опасности независимо от того, насколько неуязвимый алгоритм шифрования вы выбрали.

Важно определить, как именно вы хотите обезопасить свои данные. Шифрование обеспечивает более надежную защиту, но оно сложнее. Более простые способы кодирования скрывают данные от неискушенных надоедливых глаз, но безопасность при этом уменьшается. Не существует абсолютного шифрования или стопроцентной гарантии безопасности. Выбор соответствующего метода обеспечения безопасности означает компромисс между удобством и защищенностью. Более удобный (или недорогой в смысле вычислений) вид безопасности, как правило, обеспечивает менее надежную защиту. Иногда требуется не столько защитить данные от нескромных глаз, сколько избежать выставления конфиденциальной информации напоказ. Возможность наблюдать в форме (или в URL) содержимое поля с названием «Password» в виде обычного текста может больше обеспокоить ваших пользователей, чем те же самые данные в кодировке Base64. Рецепт 14.2 показывает, как скрывать данные с помощью алгоритма Base64.

Информация, требующая деликатного обращения, должна быть защищена не только на сервере, но и во время ее перемещения по сети между сервером и пользователями. Данные, передаваемые по обычному протоколу HTTP, доступны любому, получившему доступ к сети, в любой точке между сервером и пользователем. Рецепт 14.10 рассказывает, как использовать протокол HTTP поверх SSL, чтобы помешать сетевым шпионам подглядывать в данные во время их передачи.

Существует множество предпосылок для усиления безопасности, не имеющих отношения к технике. Назначение паролей, полученных в результате случайной перетасовки букв, цифр и символов пунктуации, – идея не самая удачная, если эти пароли настолько трудны для запоминания, что пользователи записывают их на стикерах, прикрепленных к их мониторам. Как мы уже говорили, безопасность это не аб-

солют, а компромисс между удобством и защитой. Когда вы будете применять рецепты этой главы для защиты своих данных, найдите приемлемое соотношение между риском, которому подвергаются ваши данные, и степенью неудобства, вносимого безопасностью.[1]

14.1. Не храните пароли на своем сайте

Задача

Пароль может быть необходим, например для соединения с базой данных. А хранить пароль в используемых для этого файлах PHP вы не хотите, ведь злоумышленник может получить к ним доступ.

Решение

Сохраните пароль в переменной окружения в файле, который сервер загружает при старте, а затем просто сошлитесь на нее в сценарии:

```
mysql_connect('localhost',$_ENV['MYSQL_USER'],$_ENV['MYSQL_PASSWORD']);
```

Обсуждение

Хотя эта методика исключает пароли из исходного кода ваших страниц, она делает их доступными из других мест, которые также требуют защиты. Тут очень важно обеспечить отсутствие общедоступных страниц, которые включают в себя вызов функции `phpinfo()`. Функция `phpinfo()` показывает переменные окружения, доступные в сценарии, поэтому она покажет и пароли, размещенные в таких переменных.

Далее, особенно если вы работаете в схеме с разделяемым хостингом, установите переменные окружения таким образом, чтобы они были доступны только вашему виртуальному хосту, а не всем пользователям, работающим в разделяемом пространстве хостинга. В случае работы с веб-сервером Apache это можно сделать, установив переменные в отдельном файле, а не в главном файле конфигурации:

```
SetEnv  MYSQL_USER     "susannah"
SetEnv  MYSQL_PASSWORD "y23a!t@ce8"
```

Включите этот отдельный файл в директиву `<VirtualHost>` вашего сайта в главном файле конфигурации следующим образом:

```
Include "/usr/local/apache/database-passwords"
```

Убедитесь, что этот отдельный файл, содержащий пароль (т. е. */usr/local/apache/database-passwords*), доступен для чтения только пользователю, управляющему соответствующим виртуальным хостом. Сам

[1] Книга «Practical UNIX and Internet Security» Симсона Гарфинкеля (Simson Garfinkel) и Женэ Спаффорда (Gene Spafford) (O'Reilly) предлагает несколько полезных и (что неудивительно) практических советов, как спланировать сбалансированные действия в условиях рискованного управления.

Apache обычно запускается и читает файлы конфигурации как root, поэтому он сможет прочитать включенный файл.

См. также

Документацию по директиве Apache `Include` на *http://httpd.apache.org/docs/mod/core.html#include*.

14.2. Сокрытие данных при помощи кодирования

Задача

Необходимо предотвратить просмотр данных как простого текста. Например, вы не хотите, чтобы кто-нибудь увидел секретную информацию формы, просмотрев исходный код страницы.

Решение

Закодируйте данные с помощью функции `base64_encode()`:

```
$personal_data = array('code' => 5123, 'blood_type' => 'O');
$info = base64_encode(serialize($personal_data));
print '<input type="hidden" name="info" value="'.$info.'">';
<input type="hidden" name="info"
value="YToyOntzOjQ6ImNvZGUiO2k6NTEyMztzOjEwOiJibG9vZF9OeXBlIjtzOjE6Ik8iO30=">
```

Декодируйте данные с помощью функции `base64_decode()`:

```
$personal_data = unserialize(base64_decode($_REQUEST['info']));
get_transfusion($personal_data['blood_type']);
```

Обсуждение

Алгоритм Base64 кодирует информацию в виде строки символов, цифр и знаков пунктуации. Это делает его идеальным средством преобразования двоичных данных в обычный текст и сокрытия данных.

См. также

Документацию по функции `base64_encode()` на *http://www.php.net/base64-encode* и по функции `base64_decode()` на *http://www.php.net/base64-decode*; определенный в RFC 2045 алгоритм Base64, который доступен на *http://www.faqs.org/rfcs/rfc2045.html*.

14.3. Проверка данных с помощью хеширования

Задача

Необходимо не дать возможности пользователям изменять данные, посланные им в cookie или в элементе формы.

Решение

Вместе с данными пошлите хеш этих данных с секретным словом, полученный в соответствии с алгоритмом MD5. При получении этих данных обратно вычислите хеш полученного значения с помощью того же самого секретного слова. Если они не совпадают, значит, данные были изменены.

Покажем, как напечатать хеш в скрытом поле формы:

```
$secret_word = 'flyingturtle';
$id = 2836;
$hash = md5($secret_word . $id);

print<<<_HTML_
<input type="hidden" name="id" value="$id">
<input type="hidden" name="idhash" value="$hash">
_HTML_;
```

Ниже показано, как проверить данные скрытого поля формы после ее получения обратно:

```
$secret_word = 'flyingturtle';

if (md5($secret_word . $_REQUEST['id']) == $_REQUEST['idhash']) {
    $id = $_REQUEST['id'];
} else {
    die("Invalid data in $_REQUEST[id]");
}
```

Обсуждение

При обработке информации представленной формы вычислите хеш представленного значения `$_REQUEST['id']` и секретного слова. Если он совпадает с представленным хешем, то значение переменной `$_REQUEST['id']` не было изменено пользователем. Если хеши не совпадают, то понятно, что принятое значение переменной `$_REQUEST['id']` не совпадает с тем значением, которое вы послали.

Чтобы использовать проверочный хеш с cookie, добавьте хеш к значению cookie с помощью функции `join()`:

```
$secret_word = 'flyingturtle';
$cookie_value = 'Ellen';
$hash = md5($secret_word . $id);

setcookie('name',join('|',array($cookie_value,$hash)));
```

Выделите хеш из значения cookie с помощью функции `explode()`:

```
$secret_word = 'flyingturtle';
list($cookie_value,$cookie_hash) = explode('|',$_COOKIE['name'],2);
if (md5($secret_word . $cookie_value) == $cookie_hash) {
    $name = $cookie_value;
} else {
```

```
    die('Invalid data in $_COOKIE[name]');
}
```

Использование хеша для проверки информации в форме или cookie, очевидно, зависит от секретного слова, использованного при вычислении хеша. Если злонамеренный пользователь вскроет ваше секретное слово, то хеш не обеспечит защиты. Помимо тщательной защиты секретного слова, целесообразно почаще менять его. Можно обеспечить дополнительный уровень защиты, если выбирать для использования в хеше различные секретные слова, применяя специфические варианты, основанные на некоторых свойствах значения переменной `$id` (например, 10 различных слов, выбранных с помощью `$id%10`). Таким образом можно помешать взлому, если одно из слов стало известным.

Если инсталлирован модуль *mhash*, то вы не ограничены хешами типа MD5. Модуль *mhash* поддерживает несколько алгоритмов хеширования. Более подробную информацию о модуле *mhash* можно найти в разделе *mhash* в оперативной справке по PHP или на домашней страничке модуля *mhash* на *http://mhash.sourceforge.net/*.

См. также

Рецепт 8.10, в котором проверочный хеш применяется для основанной на cookie аутентификации пользователей; пример применения хешей со скрытыми переменными формы в рецепте 9.3; документацию по функции `md5()` на *http://www.php.net/md5* и по расширению *mhash* на *http://www.php.net/mhash*.

14.4. Хранение паролей

Задача

Необходимо сохранять пароли пользователей, чтобы они могли заходить на ваш веб-сайт.

Решение

Когда пользователь регистрируется, зашифруйте выбранный им пароль с помощью функции `crypt()` и сохраните зашифрованный пароль в базе данных пользователей:

```
// шифруем пароль
$encrypted_password = crypt($_REQUEST['password']);

// заносим переменную $encrypted_password в базу данных пользователей
$dbh->query('INSERT INTO users (username,password) VALUES (?,?)',
            array($_REQUEST['username'],$encrypted_password));
```

Затем, когда пользователь попытается войти на ваш веб-сайт, зашифруйте представленный им пароль с помощью функции `crypt()` и сравните его с сохраненным и зашифрованным ранее паролем. Если два за-

шифрованных значения совпадают, значит, пользователь предоставил верный пароль:

```
$encrypted_password =
    $dbh->getOne('SELECT password FROM users WHERE username = ?',
                 array($_REQUEST['username']));

if (crypt($_REQUEST['password'],$encrypted_password) == $encrypted_password) {
  // успешный вход
} else {
  // неудачный вход
}
```

Обсуждение

Хранение паролей в зашифрованном виде предотвращает раскрытие пользовательских учетных записей, если кто-нибудь получит несанкционированный доступ к имени пользователя и паролю в вашей базе данных. (Хотя такие неавторизованные доступы могут предвещать и другие проблемы безопасности.)

После первого шифрования пароля функция `crypt()` предоставляет два случайным образом сгенерированных базовых символа, которые помещаются в начало зашифрованного пароля. Передача функции `crypt()` переменной `$encrypted_password` при проверке предоставленного пользователем пароля заставляет функцию `crypt()` снова использовать те же самые символы для базиса шифрования. Случайный базис уменьшает уязвимость к словарным атакам, в которых сравниваются зашифрованные пароли с зашифрованными значениями общеупотребительных слов. Тем не менее имеет смысл помешать пользователям выбирать в качестве паролей простые слова или другие легкие для взлома комбинации символов. Рецепт 14.5 предоставляет функцию для фильтрования легко угадываемых паролей.

Функция `crypt()` использует односторонний алгоритм шифрования. Это означает, что в настоящее время невозможно (или, по крайней мере, непомерно дорого с вычислительной точки зрения) превратить сгенерированный функцией `crypt()` зашифрованный текст обратно в простой текст. Это делает сохраненные пароли более защищенными, но это также означает, что вам не удастся получить пользовательские пароли в легко читаемом виде, даже если это будет необходимо. Поэтому, если пользователь, к примеру, забыл свой пароль, вы не сможете ему сообщить этот же самый пароль. Лучшее, что вы сможете сделать, – это установить новое значение пароля и сообщить пользователю его новый пароль. Способу работы с потерянными паролями посвящен рецепт 14.6.

См. также

Информацию о хранении зашифрованных данных в рецепте 14.8; документацию по функции `crypt()` на *http://www.php.net/crypt*.

14.5. Проверка надежности пароля

Задача

Необходимо обеспечить, чтобы пользователи выбирали пароли, трудные для угадывания перебором.

Решение

Проверьте выбранный пользователем пароль с помощью функции `pc_passwordcheck()`, показанной далее в примере 14.1. Например:

```
if ($err = pc_passwordcheck($_REQUEST['username'],$_REQUEST['password'])) {
    print "Bad password: $err";
    // Предлагаем пользователю выбрать другой пароль
}
```

Обсуждение

Функция `pc_passwordcheck()`, показанная в примере 14.1, выполняет определенные проверки выбранных пользователем паролей на предмет легкости их угадывания. Она возвращает строку с описанием проблемы, если пароль не отвечает критерию. Пароль должен иметь длину, равную минимум 6 символам, и быть составленным из букв верхнего и нижнего регистра, цифр и специальных символов. Пароль не может содержать имя пользователя в обычном или обратном порядке следования символов. Кроме того, пароль не должен содержать слов из словаря. Имя файла для списка слов, используемых при словарной проверке, находится в переменной `$word_file`.

Проверки пароля на имя пользователя или на слова из словаря применяются и к вариантам пароля, в которых буквы заменены похожими на них цифрами. Например, если предоставлен пароль `w0rd$%`, то функция также проверяет на имя пользователя и словарные слова строку `word$%`. Символ «0» превращается в символ «o». Аналогично «5» превращается в «s», «3» в «e», а «1» и «!» в «l» (el).

Пример 14.1. pc_passwordcheck( )

```
function pc_passwordcheck($user,$pass) {
    $word_file = '/usr/share/dict/words';

    $lc_pass = strtolower($pass);
    // также проверяем пароль с цифрами и знаками пунктуации,
    // заменяющими буквы
    $denum_pass = strtr($lc_pass,'5301!','seoll');
    $lc_user = strtolower($user);

    // пароль должен состоять минимум из шести символов
    if (strlen($pass) < 6) {
        return 'The password is too short.';
    }
```

```
// пароль не может быть именем пользователя
// (или перевернутым именем пользователя)
if (($lc_pass == $lc_user) || ($lc_pass == strrev($lc_user)) ||
    ($denum_pass == $lc_user) || ($denum_pass == strrev($lc_user))) {
    return 'The password is based on the username.';
}

// подсчитываем, сколько в пароле символов верхнего
// и нижнего регистра и цифр
$uc = 0; $lc = 0; $num = 0; $other = 0;
for ($i = 0, $j = strlen($pass); $i < $j; $i++) {
    $c = substr($pass,$i,1);
    if (preg_match('/^[[:upper:]]$/',$c)) {
        $uc++;
    } elseif (preg_match('/^[[:lower:]]$/',$c)) {
        $lc++;
    } elseif (preg_match('/^[[:digit:]]$/',$c)) {
        $num++;
    } else {
        $other++;
    }
}

// в пароле должно быть более двух символов,
// по крайней мере, двух различных типов
$max = $j - 2;
if ($uc > $max) {
    return "The password has too many upper case characters.";
}
if ($lc > $max) {
    return "The password has too many lower case characters.";
}
if ($num > $max) {
    return "The password has too many numeral characters.";
}
if ($other > $max) {
    return "The password has too many special characters.";
}

// пароль не должен содержать слов из словаря
if (is_readable($word_file)) {
    if ($fh = fopen($word_file,'r')) {
        $found = false;
        while (! ($found || feof($fh))) {
            $word = preg_quote(trim(strtolower(fgets($fh,1024))),'/');
            if (preg_match("/$word/",$lc_pass) ||
                preg_match("/$word/",$denum_pass)) {
                $found = true;
            }
        }
        fclose($fh);
        if ($found) {
            return 'The password is based on a dictionary word.';
```

```
                }
            }
        }

        return false;
    }
```

См. также

Полезные указания по выбору пароля, доступные на *http://tns.sdsu.edu/security/passwd.html*.

14.6. Работа с потерянными паролями

Необходимо предоставить пароль пользователю, заявившему о его утрате.

Решение

Сгенерируйте новый пароль и пошлите его по адресу электронной почты пользователя (предварительно сохраненному вами в файле):

```
// генерируем новый пароль
$new_password = '';
$i = 8;
while ($i--) { $new_password .= chr(mt_rand(33,126)); }

// зашифровываем новый пароль
$encrypted_password = crypt($new_password);

// сохраняем новый зашифрованный пароль в базе данных
$dbh->query('UPDATE users SET password = ? WHERE username = ?',
            array($encrypted_password,$username));

// шлем письмо пользователю с паролем в виде обычного текста
mail($email,"New Password","Your new password is $new_password");
```

Обсуждение

Если пользователь забывает свой пароль, а пароли хранятся в соответствии с рекомендациями рецепта 14.4, то невозможно предоставить забытый пароль. Односторонняя природа функции `crypt()` не позволяет восстановить исходный пароль.

Вместо этого генерируется новый пароль и посылается по существующему адресу. Если вы посылаете новый пароль по адресу, которого нет в файле данного пользователя, то у вас нет способа проверить, действительно ли этот адрес принадлежит указанному пользователю. Это может быть попыткой атаки с целью получения пароля и имитацией реального пользователя.

Поскольку почтовое сообщение с новым паролем не зашифровано, то программа в разделе «Решение» не включает в него имя пользователя, чтобы как-то уменьшить вероятность использования украденного па-

роля атакующим, перехватившим почтовое сообщение. Чтобы совсем исключить обнаружение пароля в письме, разрешите пользователю идентифицировать себя без пароля, отвечая на один или более личных вопросов (ответы на которые находятся в файле пользователя). Вопросы могут быть типа: «Как звали вашего первого питомца?», или «Какую фамилию носила ваша мать в девичестве?», или еще что-нибудь такое, что злонамеренный атакующий вряд ли знает. Если пользователь дает правильные ответы на ваши вопросы, то можно позволить ему указать новый пароль.

Один из способов добиться компромисса между безопасностью и читаемостью состоит в том, чтобы генерировать пароль пользователя, состоящий из реальных слов, перемежающихся числами.

```
$words =
array('dished','mother','basset','detain','sudden','fellow','logged','sonora',
     'earths','remove','dustin','snails','direct','serves','daring','cretan',
     'chirps','reward','snakes','mchugh','uphold','wiring','gaston','nurses',
     'regent','ornate','dogmas','singed','mended','hinges','latent','verbal',
     'grimes','ritual','drying','hobbes','chests','newark','sourer','rumple');

mt_srand((double) microtime() * 1000000);
$word_count = count($words);

$password = sprintf('%s%02d%s', $words[mt_rand(0,$word_count - 1)],
                    mt_rand(0,99), $words[mt_rand(0,$word_count - 1)]);

print $password;
```

Этот фрагмент программы генерирует пароли из двух шестибуквенных слов с двумя цифрами между ними, наподобие `mother43hinges` или `verbal08chirps`. Эти пароли длинные, но слова в них делают пароли легкими для запоминания.

См. также

Рецепт 14.4 о хранении зашифрованных паролей и рецепт 14.5, где более подробно описывается проверка надежности пароля.

14.7. Шифрование и дешифрование данных

Задача

Необходимо зашифровать и расшифровать информацию с помощью одного из множества популярных алгоритмов.

Решение

Используйте расширение PHP *mcrypt*:

```
$key  = 'That golden key that opens the palace of eternity.';
$data = 'The chicken escapes at dawn. Send help with Mr. Blue.';
$alg  = MCRYPT_BLOWFISH;
```

```
$mode = MCRYPT_MODE_CBC;

$iv = mcrypt_create_iv(mcrypt_get_iv_size($alg,$mode),MCRYPT_DEV_URANDOM);
$encrypted_data = mcrypt_encrypt($alg, $key, $data, $mode, $iv);
$plain_text = base64_encode($encrypted_data);

print $plain_text."\n";
$decoded = mcrypt_decrypt($alg,$key,base64_decode($plain_text),$mode,$iv);
print $decoded."\n";
NNB9WnuCYjyd3Y7vUh7XDfWFCWnQY0BsMehHNmBHbGOdJ3cM+yghABb/ XyrJ+w3xz9tms74/a70=
The chicken escapes at dawn. Send help with Mr. Blue.
```

Обсуждение

Расширение *mcrypt* представляет собой интерфейс библиотеки *mcrypt*, включающей множество различных алгоритмов шифрования. Данные шифруются и расшифровываются с помощью функций `mcrypt_encrypt()` и `mcrypt_decrypt()` соответственно. Каждая их них принимает пять аргументов. Первый аргумент представляет применяемый алгоритм. Чтобы определить, какие алгоритмы поддерживает библиотека *mcrypt* в вашей системе, вызовите функцию `mcrypt_list_algorithms()`. Полный список алгоритмов библиотеки *mcrypt* представлен в табл. 14.1. Второй аргумент – это ключ шифрования, а третий – данные для шифрования или расшифровки. Четвертый аргумент устанавливает режим шифрования или расшифровки (список режимов возвращает функция `mcrypt_list_modes()`). Пятый аргумент – это вектор инициализации (IV), используемый некоторыми режимами в качестве составляющей процесса шифрования или дешифрования.

В табл. 14.1 перечислены все возможные алгоритмы библиотеки *mcrypt*, в том числе постоянные значения, которыми обозначаются алгоритмы, ключ и размеры блоков в битах, а также информация о том, поддерживаются ли алгоритмы версиями 2.2.x и 2.4.x библиотеки *libmcrypt*.

Таблица 14.1. Константы алгоритмов библиотеки mcrypt

Константа алгоритма	Описание	Размер ключа	Размер блока	2.2. x	2.4. x
MCRYPT_3DES	Тройной DES	168 (112 эффективный)	64	Да	Да
MCRYPT_TRIPLEDES	Тройной DES	168 (112 эффективный)	64	Нет	Да
MCRYPT_3WAY	3way (Жоан Демен – Joan Daemen)	96	96	Да	Нет
MCRYPT_THREEWAY	3way	96	96	Да	Да
MCRYPT_BLOWFISH	Blowfish (Брюс Шнайер – Bruce Schneier)	До 448	64	Нет	Да

Константа алгоритма	Описание	Размер ключа	Размер блока	2.2. x	2.4. x
MCRYPT_BLOWFISH_COMPAT	Blowfish, совместимый с другими реализациями	До 448	64	Нет	Да
MCRYPT_BLOWFISH_128	Blowfish	128	64	Да	Нет
MCRYPT_BLOWFISH_192	Blowfish	192	64	Да	
MCRYPT_BLOWFISH_256	Blowfish	256	64	Да	Нет
MCRYPT_BLOWFISH_448	Blowfish	448	64	Да	Нет
MCRYPT_CAST_128	CAST (Карлайл Адамс и Стэффорд Таварес – Carlisle Adams и Stafford Tavares)	128	64	Да	Да
MCRYPT_CAST_256	CAST	256	128	Да	Да
MCRYPT_CRYPT	Однопроходный алгоритм шифрования Unix	104	8		Да
MCRYPT_ENIGNA	Однопроходный алгоритм шифрования Unix	104	8	Нет	Да
MCRYPT_DES	Стандарт шифрования данных США (U.S. Data Encryption Standard)	56	64	Да	Да
MCRYPT_GOST	Государственный стандарт СССР	256	64	Да	Да
MCRYPT_IDEA	Международный стандарт шифрования данных (International Data Encryption Algorithm)	128	64	Да	Да
MCRYPT_LOKI97	LOKI97 (Лоури Браун, Джозеф Пипрзик – Lawrie Brown, Josef Pieprzyk)	128, 192 или 256	64	Да	Да
MCRYPT_MARS	MARS (IBM)	128– 448	128	Нет	Да
MCRYPT_PANAMA	PANAMA (Жоан Демен, Крэйг Клапп – Joan Daemen, Craig Clapp)	–	Поток	Нет	Да
MCRYPT_RC2	Rivest Cipher 2	8–1024	64	Нет	Да
MCRYPT_RC2_1024	Rivest Cipher 2	1024	64	Да	Нет
MCRYPT_RC2_128	Rivest Cipher 2	128	64	Да	Нет
MCRYPT_RC2_256	Rivest Cipher 2	256	64	Да	Нет

Таблица 14.1 (продолжение)

Константа алгоритма	Описание	Размер ключа	Размер блока	2.2. x	2.4. x
MCRYPT_RC4	Rivest Cipher 4	До 2048	Поток	Да	Нет
MCRYPT_ARCFOUR	Не имеющий торговой марки, RC4-совместимый	До 2048	Поток	Нет	Да
MCRYPT_ARCFOUR_IV	Arcfour с вектором инициализации	До 2048	Поток	Нет	Да
MCRYPT_RC6	Rivest Cipher 6	128, 192 или 256	128	Нет	Да
MCRYPT_RC6_128	Rivest Cipher 6	128	128	Да	Нет
MCRYPT_RC6_192	Rivest Cipher 6	192	128	Да	Нет
MCRYPT_RC6_256	Rivest Cipher 6	256	128	Да	Нет
MCRYPT_RIJNDAEL_128	Rijndael (Жоан Демен, Винсент Риджмен – Joan Daemen, Vincent Rijmen)	128	128	Да	Да
MCRYPT_RIJNDAEL_192	Rijndael	192	192	Да	Да
MCRYPT_RIJNDAEL_256	Rijndael	256	256	Да	Да
MCRYPT_SAFERPLUS	SAFER+ (основанный на SAFER)	128, 192 или 256	128	Да	Да
MCRYPT_SAFER_128	Безопасная и быстрая процедура шифрования с улучшенным набором усиленных ключей (Secure And Fast Encryption Routine with strengthened key schedule)	128	64	Да	Да
MCRYPT_SAFER_64	Безопасная и быстрая процедура шифрования с усиленным ключом (Secure And Fast Encryption Routine with strengthened key)	64	64	Да	Да
MCRYPT_SERPENT	Serpent (Росс Андерсон, Эли Бихэм, Ларс Надсен – Ross Anderson, Eli Biham, Lars Knudsen)	128, 192 или 256	128	No	Да
MCRYPT_SERPENT_128	Serpent	128	128	Да	Нет
MCRYPT_SERPENT_192	Serpent	192	128	Да	Нет

Константа алгоритма	Описание	Размер ключа	Размер блока	2.2. x	2.4. x
MCRYPT_SERPENT_256	Serpent	256	128	Да	Нет
MCRYPT_SKIPJACK	Стандарт шифрования Управления национальный безопасности США (U.S. NSA Clipper Escrowed Encryption Standard)	80	64	Нет	Да
MCRYPT_TWOFISH	Twofish (Counterpane Systems)	128, 192 или 256	128	Нет	Да
MCRYPT_TWOFISH_128	Twofish	128	128	Да	Нет
MCRYPT_TWOFISH_192	Twofish	192	128	Да	Нет
MCRYPT_TWOFISH_256	Twofish	256	128	Да	Нет
MCRYPT_WAKE	Шифрование слов с авто-ключом, Дэвид Вилер (Word Auto Key Encryption, David Wheeler)	256	32	Нет	Да
MCRYPT_XTEA	Расширенный малый алгоритм шифрования, Дэвид Вилер, Роджер Нидхэм (Extended Tiny Encryption Algorithm, David Wheeler, Roger Needham)	128	64	Да	Да

За исключением шифруемой и дешифруемой информации, все остальные аргументы должны быть одинаковыми как при шифровании, так и при расшифровке. При использовании режима, требующего вектор инициализации, можно передать этот вектор в открытом виде вместе с зашифрованным текстом.

Различные режимы соответствуют разным обстоятельствам. Режим цепочки кодовых блоков (Cipher Block Chaining – CBC) кодирует данные по блокам и использует зашифрованное значение каждого блока (как и ключа) для вычисления зашифрованного значения следующего блока. Вектор инициализации оказывает влияние на первый блок. Режимы обратной кодовой связи (Cipher Feedback – CFB) и выходной обратной связи (Output Feedback – OFB) также используют вектор инициализации, но они шифруют данные частями меньшей длины, чем размер блока. Обратите внимание, что режим OFB небезопасен, если шифровать данные порциями, меньшими, чем блок. Режим электронной кодовой книги (Electronic Code Book – ECB) шифрует данные блоками, не зависящими друг от друга. Кроме того, режим ECB не использует векторы инициализации. Он также менее безопасен, чем дру-

гие методы, если его применять регулярно, поскольку для одного и того же исходного текста с данным ключом он всегда генерирует тот же самый зашифрованный текст. Константы для установки каждого из режимов перечислены в табл. 14.2.

Таблица 14.2. Константы режимов библиотеки mcrypt Различные

Константа режима	Описание
MCRYPT_MODE_ECB	Режим электронной кодовой книги (ECB)
MCRYPT_MODE_CBC	Режим цепочки кодовых блоков (CBC)
MCRYPT_MODE_CFB	Режим обратной кодовой связи (CFB)
MCRYPT_MODE_OFB	Режим выходной обратной связи (OFB) с 8-битной обратной связью
MCRYPT_MODE_NOFB	Режим выходной обратной связи (OFB) с *n*-битной обратной связью, где *n* – это размер блока, используемый алгоритмом (только для библиотеки *libmcrypt* версии 2.4 и выше)
MCRYPT_MODE_STREAM	Режим потокового кода (Stream Cipher) для таких алгоритмов, как RC4 и WAKE (только библиотека *libmcrypt* версии 2.4 и выше)

алгоритмы имеют разные размеры блока. Размер блока отдельного алгоритма можно определить с помощью функции mcrypt_ get_block_size(). *Точно так же длина вектора инициализации определяется алгоритмом и режимом. Функции* mcrypt_create_iv() *и* mcrypt_ get_iv_size() *облегчают создание соответствующего случайного вектора инициализации:*

```
$iv = mcrypt_create_iv(mcrypt_get_iv_size($alg,$mode),MCRYPT_DEV_URANDOM);
```

Первый аргумент функции mcrypt_create_iv() – длина вектора, а второй – это источник случайности. Имеется три варианта источника случайности: MCRYPT_DEV_RANDOM читает из псевдоустройства */dev/random*, MCRYPT_DEV_URANDOM читает из псевдоустройства */dev/urandom*, а MCRYPT_RAND использует встроенный генератор случайных чисел. Не все операционные системы поддерживают псевдоустройства случайных генераторов. Не забудьте вызвать функцию srand() перед использованием MCRYPT_RAND, для того чтобы получить неповторяющуюся последовательность случайных чисел.

Код и примеры этого рецепта совместимы с *mcrypt* 2.4. Интерфейс mcrypt в PHP поддерживает библиотеки *mcrypt* 2.2 и *mcrypt* 2.4, но между ними есть некоторые различия. В случае библиотеки *mcrypt* 2.2 PHP поддерживает только следующие *mcrypt*-функции: mcrypt_ecb(), mcrypt_cbc(), mcrypt_cfb(), mcrypt_ofb(), mcrypt_get_key_size(), mcrypt_get_block_size(), mcrypt_get_cipher_name() и mcrypt_create_iv(). Чтобы зашифровать и расшифровать данные с помощью библиотеки *mcrypt* 2.2,

вызовите соответствующую функцию `mcrypt_MODE()` на базе выбранного вами режима, и передайте ей аргумент, определяющий шифрование или дешифрование. Следующий фрагмент представляет собой версию программы из раздела «Решение», совместимую с *mcrypt* 2.2:

```
$key  = 'That golden key that opes the palace of eternity.';
$data = 'The chicken escapes at dawn. Send help with Mr. Blue.';
$alg = MCRYPT_BLOWFISH;

$iv = mcrypt_create_iv(mcrypt_get_block_size($alg),MCRYPT_DEV_URANDOM);
$encrypted_data = mcrypt_cbc($alg,$key,$data,MCRYPT_ENCRYPT);
$plain_text = base64_encode($encrypted_data);

print $plain_text."\n";

$decoded = mcrypt_cbc($alg,$key,base64_decode($plain_text),MCRYPT_DECRYPT);

print $decoded."\n";
```

См. также

Документацию по расширению *mcrypt* на *http://www.php.net/mcrypt*; библиотеку *mcrypt*, доступную на *http://mcrypt.hellug.gr/*; выбор соответствующего алгоритма и его безопасное применение требует тщательного обдумывания и планирования; более подробную информацию о библиотеке *mcrypt* и алгоритмах кодирования, которые она использует, можно найти в разделе *mcrypt* оперативной справки PHP, на домашней странице *mcrypt* и на странице программы man в */dev/random* and */dev/urandom*; в число хороших книг по криптографии входят «Applied Cryptography» Брюса Шнейера (Bruce Schneier) (Wiley) и «Cryptography: Theory and Practice» Дугласа Р. Стинсона (Douglas R. Stinson) (Chapman & Hall).[1]

14.8. Хранение зашифрованных данных в файле или базе данных

Задача

Требуется сохранить зашифрованные данные, которые в дальнейшем будут затребованы и расшифрованы вашим веб-сервером.

Решение

Вместе с зашифрованными данными сохраните дополнительную информацию, необходимую для расшифровки (такую как алгоритм, режим кодировки и вектор инициализации), но не ключ:

[1] Шнайер Б. «Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си». – Пер. с англ. – М.: Триумф, 2002.

```
// шифруем данные
$alg  = MCRYPT_BLOWFISH;
$mode = MCRYPT_MODE_CBC;
$iv = mcrypt_create_iv(mcrypt_get_iv_size($alg,$mode),MCRYPT_DEV_URANDOM);
$ciphertext = mcrypt_encrypt($alg,$_REQUEST['key'],
                             $_REQUEST['data'],$mode,$iv);

// записываем зашифрованные данные
$dbh->query('INSERT INTO noc_list (algorithm,mode,iv,data)
                         values (?,?,?,?)',
            array($alg,$mode,$iv,$ciphertext));
```

Для расшифровки данных получите от пользователя ключ и примените его к сохраненным данным:

```
$row = $dbh->getRow('SELECT * FROM noc_list WHERE id = 27');
$plaintext = mcrypt_decrypt($row->algorithm,$_REQUEST['key'],$row->data,
                            $row->mode,$row->iv);
```

Обсуждение

Программа *save-crypt.php*, показанная в примере 14.2, записывает зашифрованные данные в файл.

Пример 14.2. save-crypt.php

```
function show_form() {
    print<<<_FORM_
<form method="post" action="$_SERVER[PHP_SELF]">
<textarea name="data" rows="10" cols="40">
Enter data to be encrypted here.
</textarea>
<br>
Encryption Key: <input type="text" name="key">
<input name="submit" type="submit" value="save">
</form>
_FORM_;
}

function save_form() {
    $alg  = MCRYPT_BLOWFISH;
    $mode = MCRYPT_MODE_CBC;

    // шифруем данные
    $iv = mcrypt_create_iv(mcrypt_get_iv_size($alg,$mode),MCRYPT_DEV_URANDOM);
    $ciphertext = mcrypt_encrypt($alg, $_REQUEST['key'],
                                 $_REQUEST['data'], $mode, $iv);

    // записываем зашифрованные данные
    $filename = tempnam('/tmp','enc') or die($php_errormsg);
    $fh = fopen($filename,'w')        or die($php_errormsg);
    if (false === fwrite($fh,$iv.$ciphertext)) {
        fclose($fh);
        die($php_errormsg);
```

```
    }
    fclose($fh)                        or die($php_errormsg);

    Return $filename;
}

if ($_REQUEST['submit']) {
    $file = save_form();
    print "Encrypted data saved to file: $file";
} else {
    show_form();
}
```

Пример 14.3 показывает соответствующую программу, *get-crypt.php*, которая принимает имя файла и ключ и выдает расшифрованные данные.

Пример 14.3. get-crypt.php

```
function show_form() {
    print<<<_FORM_
<form method="post" action="$_SERVER[PHP_SELF]">
Encrypted File: <input type="text" name="file">
<br>
Encryption Key: <input type="text" name="key">
<input name="submit" type="submit" value="display">
</form>
_FORM_;
}

function display() {
    $alg  = MCRYPT_BLOWFISH;
    $mode = MCRYPT_MODE_CBC;

    $fh = fopen($_REQUEST['file'],'r') or die($php_errormsg);
    $iv = fread($fh,mcrypt_get_iv_size($alg,$mode));
    $ciphertext = fread($fh,filesize($_REQUEST['file']));
    fclose($fh);

    $plaintext = mcrypt_decrypt($alg,$_REQUEST['key'],$ciphertext,$mode,$iv);
    print "<pre>$plaintext</pre>";
}

if ($_REQUEST['submit']) {
    display();
} else {
    show_form();
}
```

Эти две программы имеют свои собственные алгоритмы шифрования и режимы, жестко в них зашитые, поэтому нет необходимости сохранять эту информацию в файле. Файл состоит из вектора инициализации, за которым сразу следуют шифрованные данные. Специальный разделитель после вектора инициализации (IV) не требуется, посколь-

ку функция `mcrypt_get_iv_size()` возвращает ровно столько байт, сколько требуется программе шифрования, чтобы прочитать весь вектор IV. Все, что находится после вектора, является зашифрованными данными.

Метод шифрования файла, реализованный в данном рецепте, предлагает защиту на случай, если атакующий получает доступ к серверу, на котором хранится файл. Без соответствующего ключа или огромных вычислительных затрат атакующий не сможет прочитать файл. Однако безопасность, которую обеспечивает этот зашифрованный файл, находится под угрозой, если данные, которые надо зашифровать, и ключ шифрования передаются между вашим сервером и броузерами пользователей в открытом виде. Любой, кто перехватывает или отслеживает сетевой трафик, получит возможность увидеть данные до того, как они будут зашифрованы. Чтобы предотвратить такой способ перехвата сообщений, следует использовать SSL.

Дополнительный риск на время шифрования информации вашим веб-сервером зависит от того, насколько легко просмотреть эти данные до того, как они будут зашифрованы и записаны в файл. Любой, кто получит доступ к серверу с правами root или администратора, может просмотреть память, используемую процессом сервера, и увидеть незашифрованные данные и ключ. Если операционная система использует файл подкачки, записывая образ памяти серверного процесса на диск, то к незашифрованным данным можно также получить доступ через своп-файл. Атаку такого типа трудно отразить, и она может быть опустошительной. После того как информация записана в файл, ее не сможет прочитать даже взломщик с правами root к веб-серверу, но если атакующий доберется до информации раньше, чем она будет записана в файл, то шифрование будет слабой защитой.

См. также

Рецепт 14.10, в котором обсуждается SSL и защита информации во время ее перемещения по сети; документацию по функции `mcrypt_encrypt()` на *http://www.php.net/mcrypt-encrypt*, по функции `mcrypt_decrypt()` на *http://www.php.net/mcrypt-decrypt*, по функции `mcrypt_create_iv()` на *http://www.php.net/mcrypt-create-iv* и по функции `mcrypt_get_iv_size()` на *http://www.php.net/mcrypt-get-iv-size*.

14.9. Совместное использование зашифрованных данных с другим веб-сайтом

Задача

Необходимо организовать безопасный обмен данными с другим веб-сайтом.

Решение

Если другой веб-сайт получает информацию с вашего веб-сайта, то поместите эти данные на страницу, защищенную паролем. Можно также предоставлять информацию в зашифрованном виде, с паролем или без него. Если необходимо доставить данные на другой веб-сервер, то передавайте потенциально шифруемые данные с помощью метода POST на защищенный паролем URL.

Обсуждение

Следующая страница требует имя пользователя и пароль, а затем кодирует и показывает содержимое файла, содержащего информацию о вчерашней активности учетной записи:

```
$user = 'bank';
$password = 'fas8uj3';

if (! (($_SERVER['PHP_AUTH_USER'] == $user) &&
       ($_SERVER['PHP_AUTH_PW'] == $password))) {
    header('WWW-Authenticate: Basic realm="Secure Transfer"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You must supply a valid username and password for access.";
    exit;
}

header('Content-type: text/plain');
$filename = strftime('/usr/local/account-activity.%Y-%m-%d',time() - 86400);
$data = join('',file($filename));

$alg  = MCRYPT_BLOWFISH;
$mode = MCRYPT_MODE_CBC;
$key  = "There are many ways to butter your toast.";

// шифруем данные
$iv = $iv = mcrypt_create_iv(mcrypt_get_iv_size($alg,$mode),
                             MCRYPT_DEV_URANDOM);
$ciphertext = mcrypt_encrypt($alg, $key, $data, $mode, $iv);

print base64_encode($iv.$ciphertext);
```

Ниже приведен соответствующий фрагмент кода для получения зашифрованной страницы и расшифровки информации:

```
$user = 'bank';
$password = 'fas8uj3';
$alg  = MCRYPT_BLOWFISH;
$mode = MCRYPT_MODE_CBC;
$key  = "There are many ways to butter your toast.";

$fh = fopen("http://$user:$password@bank.example.com/accounts.php",'r')
      or die($php_errormsg);
$data = '';
while (! feof($fh)) { $data .= fgets($fh,1048576); }
```

```
fclose($fh) or die($php_errormsg);
$binary_data = base64_decode($data);
$iv_size = mcrypt_get_iv_size($alg,$mode);
$iv = substr($binary_data,0,$iv_size);
$ciphertext = substr($binary_data,$iv_size,strlen($binary_data));

print mcrypt_decrypt($alg,$key,$ciphertext,$mode,$iv);
```

Запрашивающая программа выполняет те же шаги, что и программа шифрования, но в обратном порядке. Она извлекает зашифрованные данные, закодированные алгоритмом кодирования Base64, передавая имя пользователя и пароль. Затем она декодирует данные с помощью алгоритма Base64 и выделяет вектор инициализации. Наконец, она расшифровывает данные и печатает их.

В предыдущем примере имя пользователя и пароль посылается по сети открытым текстом, пока не будет установлено SSL-соединение. Однако если вы используете SSL, то, возможно, нет необходимости шифровать содержимое файла. Мы включили в этот пример и запрос пароля и шифрование файла, чтобы показать, как это можно сделать.

Но есть одна ситуация, когда полезна и парольная защита и шифрование файла: если файл не расшифровывается автоматически при получении. Автоматическая программа может получить зашифрованный файл и сохранить его в зашифрованном виде для последующей обработки. Таким образом, ключ для дешифрования запрашивающей программе не требуется.

См. также

Рецепт 8.9 об использовании базовой аутентификации HTTP; рецепт 14.10, в котором обсуждается SSL и защита информации во время ее перемещения по сети; документацию по функции `mcrypt_encrypt()` на *http://www.php.net/mcrypt-encrypt* и по функции `mcrypt_decrypt()` на *http://www.php.net/mcrypt-decrypt*.

14.10. Обнаружение SSL-соединения

Задача

Необходимо узнать, использует ли прибывший запрос SSL.

Решение

Проверьте значение переменной `$_SERVER['HTTPS']`:

```
if ('on' == $_SERVER['HTTPS']) {
  print "The secret ingredient in Coca-Cola is Soylent Green.";
} else {
  print "Coca-Cola contains many delicious natural and artificial flavors.";
}
```

Обсуждение

SSL действует на более низком уровне, чем HTTP. Веб-сервер и броузер обмениваются информацией через соответствующее безопасное соединение, основанное на их возможностях, и сообщения HTTP передаются через это безопасное соединение. Для взломщика, перехватывающего трафик, они представляют просто поток бессмысленных байтов, которые невозможно прочитать.

Разные веб-серверы предъявляют различные требования к использованию SSL, поэтому посмотрите документацию к вашему веб-серверу для уточнения деталей. Чтобы работать с SSL, в PHP ничего менять не нужно.

Кроме изменения программы на основании значения переменной `$_SERVER['HTTPS']`, можно также потребовать, чтобы и значения cookies изменялись только через SSL-соединение. Если последний аргумент функции `setcookie()` равен `1`, то броузер будет передавать cookie обратно на сервер только через безопасное соединение:

```
/* устанавливаем SSL-только cookie с именем "sslonly" в значение "yes",
   которое теряет силу по окончании текущей сессии броузера */
setcookie('sslonly','yes','','/','sklar.com',1);
```

Хотя броузер посылает эти cookies обратно на сервер только через SSL-соединение, сам сервер посылает их броузеру (когда вы вызываете функцию `setcookie()` на вашей странице) независимо от того, использует SSL-соединение или нет запрос страницы, которая устанавливает cookie. Если вы помещаете секретные данные в cookie, проверьте, что вы устанавливаете cookie только с помощью SSL-запроса. Не забудьте также, что информация в cookie расшифровывается на компьютере пользователя.

См. также

Рецепт 8.1, в котором обсуждается установка cookies; документацию по функции `setcookie()` на *http://www.php.net/setcookie*.

14.11. Шифрование сообщений электронной почты с помощью GPG

Задача

Необходимо послать зашифрованное сообщение по электронной почте. Например, на вашем веб-сервере принимаются заказы, и надо отсылать письмо на фабрику с описанием заявки на обработку. Шифрование почтового сообщения предотвратит передачу по сети секретных данных в открытом виде, например таких, как номера кредитных карт.

Решение

Перед посылкой зашифруйте тело почтового сообщения с помощью программы GNU Privacy Guard (GPG):

```
$message_body = escapeshellarg($message_body);
$gpg_path     = '/usr/local/bin/gpg';
$sender       = 'web@example.com';
$recipient    = 'ordertaker@example.com';
$home_dir     = '/home/web';
$user_env     = 'web';

$cmd = "echo $msg | HOME=$home_dir USER=$user_env $gpg_path " .
       '--quiet --no-secmem-warning --encrypt --sign --armor ' .
       "--recipient $recipient --local-user $sender";

$message_body = `$cmd`;

mail($recipient,'Web Site Order',$message_body);
```

Почтовое сообщение может быть расшифровано программой GPG, Pretty Good Privacy (PGP) или дополнительным расширением к почтовому клиенту, которое содержит одну из совместимых программ.

Обсуждение

PGP – это популярная программа шифрования с открытым ключом, а GPG – это программа с открытым кодом, основанная на PGP. Поскольку программа PGP отягощена множеством патентов и проблемами контроля, зачастую легче использовать GPG.

Программа в разделе «Решение» вызывает */usr/local/bin/gpg* для шифрования сообщения, находящегося в переменной `$message_body`. Она использует закрытый ключ, принадлежащий `$sender`, и открытый ключ, принадлежащий `$recipient`. Это означает, что только `$recipient` может расшифровать почтовое сообщение, и когда он это сделает, то узнает, что сообщение пришло от `$sender`.

Установка переменных окружения `HOME` и `USER` указывает GPG, где ей искать свои настройки для шифрования: *$HOME/.gnupg/secring.gpg*. Параметры `--quiet` и `--no-secmem-warning` подавляют предупреждения GPG, которые в противном случае были бы сгенерированы и могли бы добавиться к тексту сообщения. Параметры `--encrypt` и `--sign` предписывают программе GPG зашифровать сообщение и подписать его. Шифрование сообщения закрывает его от любого, кроме адресата. Подпись добавляет информацию о том, кто составил это сообщение и когда оно было составлено. Параметр `--armor` генерирует простой текст вместо двоичного кода, что делает зашифрованное сообщение более удобным для рассылки.

Обычно закрытые ключи защищаются с помощью идентификационной фразы. Если взломщик скопирует закрытый ключ, защищенный идентификационной фразой, то он не сможет расшифровать сообще-

ния с этим закрытым ключом, пока не узнает эту идентификационную фразу. GPG запрашивает идентификационную фразу во время шифрования сообщения. Однако в этом рецепте нам не нужно, чтобы закрытый ключ имел идентификационную фразу. Если бы это было так, то веб-сайт не смог бы послать новое почтовое сообщение с заявкой без участия человека, который должен каждый раз вводить идентификационную фразу. Хранение идентификационной фразы в файле и предоставление ее программе GPG каждый раз, когда вы шифруете сообщение, не повысит уровень безопасности по сравнению с ситуацией, когда идентификационная фраза не используется с самого начала.

Недостаток применения ключа для шифрования без идентификационной фразы состоит в том, что взломщик, получивший секретный ключ, может послать поддельное письмо с заявкой вашему процессу, обрабатывающему заказы. Это управляемый риск. Поскольку заказы с самого начала могут быть представлены через веб-сайт, то начиная с этого места ложная информация может внедриться в процесс выполнения заказа. Но в качестве компенсации этого допущения такие потенциально фальшивые почтовые сообщения могут быть выявлены при помощи каких-либо дополнительных процедур по перехвату ошибочных заказов. Кроме того, если хищение ключа обнаружено и проблема, вызвавшая эту ситуацию, ликвидирована, то взломщика нетрудно блокировать, заменив закрытый ключ.

См. также

Домашнюю страницу GNU Privacy Guard на *http://www.gnupg.org/* и сайт распространения MIT PGP на *http://web.mit.edu/network/pgp.html*.

15

Графика

15.0. Введение

Библиотека GD дает возможность создавать PHP-приложения с применением динамической графики, способные показывать курсы акций, результаты голосования, производительность системы или даже создавать игры. Разумеется, это не похоже на работу с Photoshop или GIMP; например нельзя нарисовать линию движением мыши. Надо точно указать форму, размер и местоположение этой линии.

GD обладает собственным программным интерфейсом, или API, и PHP старается следовать его синтаксису и соглашению о наименовании функций. Поэтому если вы знакомы с библиотекой GD по другим языкам, таким как C или Perl, то вам будет легко использовать GD и в PHP. Если же нет, то потратьте несколько минут на ознакомление с ней и вскоре будете рисовать, как Пикассо.

Набор возможностей, предоставляемых библиотекой GD, варьируется в зависимости от того, какую версию GD вы используете и какие возможности были установлены в процессе конфигурирования. Версии GD вплоть до 1.6 поддерживали чтение и запись изображений в формате GIF, но впоследствии этот код был удален из-за неувязок с правами. Новые версии GD поддерживают форматы JPEG, PNG и WBMP. Формат PNG существенно компактнее формата GIF, поддерживает больше цветов, имеет встроенную гамма-коррекцию и поддерживается всеми основными броузерами. Таким образом, отсутствие поддержки формата GIF надо рассматривать как изменение и расширение функциональности, а не как ошибку. Дополнительную информацию о формате PNG можно найти на *http://www.libpng.org/pub/png/* или в главе 21 «PNG Format» книги Дженнифер Нидерст (Jennifer Niederst) «Web Design in a Nutshell», (O'Reilly).[1]

---

[1] Нидерст Дж. «Web-мастеринг для профессионалов». – Пер. с англ. – СПб: Питер, 2000.

GD не только поддерживает разнообразные форматы файлов, но и позволяет рисовать точки, линии, прямоугольники, многоугольники, дуги, эллипсы и окружности, любого цвета. Рецепт 15.1 посвящен фигурам с прямыми линиями, а рецепт 15.2 рассказывает о криволинейных фигурах. О том, как закрасить фигуру определенным стилем, а не сплошным цветом, вы узнаете из рецепта 15.3.

Кроме того, можно рисовать текст, используя широкое множество шрифтов, включая встроенные шрифты TrueType и PostScript Type 1. В рецепте 15.4 продемонстрированы способы обращения и получения результатов из трех основных функций рисования текста, а в рецепте 15.5 объясняется, как расположить текст в центре области отображения (canvas). Эти два рецепта, в свою очередь, образуют основу для рецепта 15.6, который объединяет шаблон изображения с данными реального времени для создания динамических изображений. Библиотека GD также позволяет создавать прозрачные изображения в форматах GIF и PNG. Задание прозрачного цвета и применение прозрачности в шаблонах обсуждается в рецепте 15.7.

Рецепт 15.8 оставляет библиотеку GD в стороне и показывает, как обезопасить обработку изображений, ограничивая доступ пользователей. Наконец, мы приведем пример приложения, собирающего воедино результаты голосования и генерирующего диаграмму, отображающую эти результаты.

Все эти возможности доступны в версии GD 1.8.4, последней стабильной версии библиотеки на момент выхода этой книги. Если у вас более ранняя версия, то вы не должны испытывать затруднений. Однако если в определенном рецепте требуется конкретная версия GD, мы будем упоминать об этом отдельно.

PHP также поддерживает и версии GD 2.x, которые на момент написания этого текста еще находятся в стадии бета-версий. Несмотря на их бета-статус, эти новые версии довольно стабильны и имеют много новых возможностей. В частности, версия 2.x поддерживает реалистичное цветовоспроизведение изображений, что позволяет библиотеке GD читать в форматах PNG и JPEG почти без потери в качестве. Кроме того, версии GD 2.x поддерживают альфа-каналы формата PNG, что дает возможность указывать уровень прозрачности для каждого пиксела.

Обе версии библиотеки GD можно загрузить с официального сайта GD на *http://www.boutell.com/gd/*. Раздел GD онлайн-руководства по PHP на *http://www.php.net/image* также перечисляет расположение дополнительных библиотек, необходимых для поддержки формата JPEG и шрифтов Type 1.

Существует два простых способа проверить, какая версия библиотеки GD, если таковая имеется, инсталлирована на вашем сервере и как она сконфигурирована. Первый способ состоит в вызове функции `phpinfo()`. Проверьте значение параметра `--with-gd` в начале вывода в разделе «Configure Command» (конфигурационная директива); затем внизу

страницы посмотрите раздел, озаглавленный «gd», в котором находится более подробная информация о том, какая версия библиотеки GD инсталлирована и какие возможности доступны. Второй способ – проверка значения, возвращаемого функцией `function_exists('imagecreate')`. Если она возвращает `true`, значит, GD инсталлирована. Функция `imagetypes()` возвращает битовое поле, показывающее, какие графические форматы доступны. Дополнительную информацию по применению этой функции можно найти на *http://www.php.net/imagetypes*. Тем же, кому понадобятся возможности, которые пока не доступны, придется самостоятельно пересобрать PHP или попросить об этом своего интернет-провайдера (ISP).

В основном процесс генерации изображения состоит из трех этапов: создания изображения, добавления графики и текста в область отображения (canvas) и отображения или записи графики. Например:

```
$image = ImageCreate(200, 50);
$background_color = ImageColorAllocate($image, 255, 255, 255); // white
$gray             = ImageColorAllocate($image, 204, 204, 204); // gray

ImageFilledRectangle($image, 50, 10, 150, 40, $gray);

header('Content-type: image/png');
ImagePNG($image);
```

Вывод этого фрагмента кода, рисующего серый прямоугольник на белом фоне, показан на рис. 15.1.

Рис. 15.1. *Серый прямоугольник на белом фоне*

Сначала создается область отображения (canvas). Функция `ImageCreate()` возвращает не само изображение, а его дескриптор. Реальную графику она не представляет до тех пор, пока вы специально не укажете PHP, что это надо сделать. Функция `ImageCreate()` позволяет манипулировать несколькими изображениями одновременно.

В качестве параметров функции `ImageCreate()` передаются ширина и высота графики в пикселах. В данном случае это 200 пикселов в ширину и 50 пикселов в высоту. Можно не только создавать новое изображение, но и редактировать уже существующие. Чтобы открыть графическое изображение в формате PNG, вызовите функцию `ImageCreateFromPNG()` или функцию с похожим названием для работы с файлом в другом графическом формате. Единственный аргумент – имя файла, а сами файлы могут находиться как на локальной машине, так и на удаленных серверах:

```
// открываем PNG на локальной машине
$graph = ImageCreateFromPNG('/path/to/graph.png');
```

```
// открываем JPEG на удаленном сервере
$icon  = ImageCreateFromJPEG('http://www.example.com/images/icon.jpeg');
```

Создав область отображения для редактирования, получаем доступ к цветам изображения с помощью вызова функции `ImageColorAllocate()`:

```
$background_color = ImageColorAllocate($image, 255, 255, 255); // white
$gray             = ImageColorAllocate($image, 204, 204, 204); // gray
```

Функция `ImageColorAllocate()` принимает дескриптор изображения, подлежащего раскраске, и три целых числа. Эти три целочисленных значения находятся в диапазоне от 0 до 255 и определяют красную, зеленую и синюю компоненты цвета. Они представляют ту же самую цветовую комбинацию RGB, которая используется в HTML для установки цвета шрифта или фона. Таким образом, белый цвет – это 255, 255, 255; черный цвет – это 0, 0, 0, а все остальные цвета располагаются между этими значениями.

Первый вызов функции `ImageAllocateColor()` устанавливает цвет фона. Дополнительные вызовы определяют цвета рисования для линий, фигур или текста. Итак, установите цвет фона, равный 255, 255, 255, а затем возьмите серое перо с помощью вызова `ImageAllocateColor($image, 204, 204, 204)`. Может показаться странным, что цвет фона определяется вызовом `ImageAllocateColor()`, а не с помощью отдельной функции. Но таков общий стиль работы библиотеки GD, и PHP соблюдает это соглашение.

Вызовите функцию `ImageFilledRectangle()` для размещения прямоугольника в области отображения. Функция `ImageFilledRectangle()` принимает несколько параметров: дескриптор области, на которой происходит рисование, координаты x и y левого верхнего угла прямоугольника, координаты x и y правого нижнего угла прямоугольника и, наконец, цвет фигуры. Приказываем функции `ImageFilledRectangle()` нарисовать серый прямоугольник в области изображения `$image`, начиная в точке (50,10) и заканчивая в точке (150,40):

```
ImageFilledRectangle($image, 50, 10, 150, 40, $gray);
```

В отличие от декартового графика, точка с координатами (0,0) это не нижний левый, а верхний левый угол. Поэтому вертикальная координата пятна, отстоящего на 10 пикселов от верхней границы области отображения высотой в 50 пикселов, равна 10, поскольку оно находится на 10 пикселов ниже верхней границы. И не равна 40, поскольку измеряется сверху вниз, а не снизу вверх. Она также не равна –10, поскольку направление вниз считается положительным направлением, а не отрицательным.

Теперь изображение полностью готово, и его можно «подавать». Сначала посылаем заголовок `Content-type`, чтобы дать возможность броузеру узнать тип посылаемого изображения. В данном случае мы посылаем PNG. Затем с помощью функции `ImagePNG()` приказываем PHP вы-

дать изображение. После посылки изображения можно считать задачу выполненной:

```
header('Content-Type: image/png');
ImagePNG($image);
```

Чтобы записать изображение на диск вместо посылки его броузеру, передайте функции `ImagePNG()` второй аргумент, указывающий, куда записать файл:

```
ImagePng($image, '/path/to/your/new/image.png');
```

Поскольку файл не передается броузеру, то нет необходимости вызывать функцию `header()`. Не забудьте указать путь и имя изображения, а также убедиться, что PHP обладает правом записи в указанный каталог.

PHP удаляет изображение по окончании сценария, но если надо вручную освободить память, занятую изображением, то с помощью вызова функции `ImageDestroy($image)` можно заставить PHP удалить изображение немедленно.

15.1. Рисование линий, прямоугольников и многоугольников

Задача

Пусть требуется нарисовать линию, прямо- или многоугольник. Кроме того, надо иметь возможность управлять типом прямоугольника или многоугольника (закрашенный или незакрашенный). Например, если необходимо нарисовать гистограмму или создать диаграмму котировки акций.

Решение

Линию можно нарисовать посредством функции `ImageLine()`:

```
ImageLine($image, $x1, $y1, $x2, $y2, $color);
```

Функция `ImageRectangle()` позволяет нарисовать незакрашенный прямоугольник:

```
ImageRectangle($image, $x1, $y1, $x2, $y2, $color);
```

А функция `ImageFilledRectangle()` — сплошной прямоугольник:

```
ImageFilledRectangle($image, $x1, $y1, $x2, $y2, $color);
```

Для того чтобы нарисовать незакрашенный многоугольник, вызовите функцию `ImagePolygon()`:

```
$points = array($x1, $y1, $x2, $y2, $x3, $y3);
ImagePolygon($image, $points, count($points)/2, $color);
```

Закрашенный многоугольник можно нарисовать, вызвав функцию `ImageFilledPolygon()`:

```
$points = array($x1, $y1, $x2, $y2, $x3, $y3);
ImageFilledPolygon($image, $points, count($points)/2, $color);
```

Обсуждение

Прототипы всех пяти функций в разделе «Решение» похожи. Первый параметр представляет собой дескриптор области отображения. Следующий набор параметров – это координаты x и y, указывающие библиотеке GD, как рисовать фигуру. В функции `ImageLine()` четыре координаты определяют концы линии, а в функции `ImageRectangle()` они указывают противоположные углы прямоугольника. Например, вызов `ImageLine($image, 0, 0, 100, 100, $color)` генерирует диагональ. Передав те же самые параметры функции `ImageRectangle()`, получим прямоугольник с углами в точках (0,0), (100,0), (0,100) и (100,100). Обе фигуры показаны на рис. 15.2.

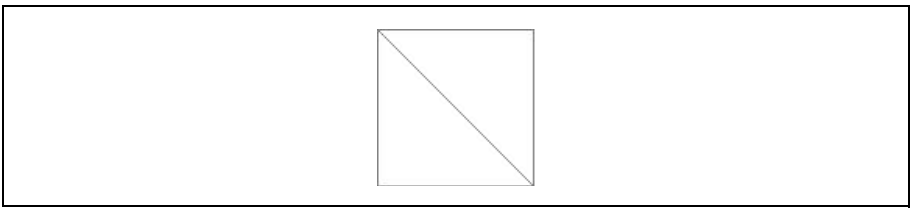

Рис. 15.2. *Диагональ и квадрат*

Функция `ImagePolygon()` немного отличается, поскольку она принимает переменное количество вершин. Поэтому вторым параметром является массив координат x и y. Функция начинает работу с первого набора точек и рисует линии от вершины к вершине до тех пор, пока не замкнет фигуру, возвратившись к начальной точке. Требуется по крайней мере три вершины многоугольника (всего шесть элементов массива). Третий параметр – это количество вершин в фигуре; поскольку это всегда половина числа элементов массива точек, то гибким значением для этого параметра является `count($points) / 2`, так как оно позволяет обновлять массив вершин, не затрагивая вызов функции `ImageLine()`.

Наконец, все функции принимают последний параметр, определяющий цвет рисования. Обычно это значение, возвращенное функцией `ImageColorAllocate()`, но оно может быть представлено константами `IMG_COLOR_STYLED` или `IMG_COLOR_STYLEDBRUSHED`, если требуется нарисовать несплошные линии, что обсуждается в рецепте 15.3.

Все эти функции рисуют незакрашенные фигуры. Чтобы заставить библиотеку GD заполнить фигуру цветом рисования, вызывайте функции `ImageFilledRectangle()` и `ImageFilledPolygon()` с тем же набором аргументов, что и для их незакрашенных собратьев.

См. также

Дополнительную информацию о рисовании других типов фигур в рецепте 15.2; рецепт 15.3, где более подробно описывается рисование с применением стилей и кистей; документацию по функции `ImageLine()` на *http://www.php.net/imageline*, по функции `ImageRectangle()` на *http://www.php.net/imagerectangle*, по функции `ImagePolygon()` на *http://www. php.net/imagepolygon* и по функции `ImageColorAllocate()` на *http://www. php.net/imagecolorallocate*.

15.2. Рисование дуг, эллипсов и окружностей

Задача

Требуется нарисовать открытые кривые и закрашенные криволинейные фигуры. Например, надо начертить секторную диаграмму, показывающую результаты голосования пользователей.

Решение

Для того чтобы нарисовать дугу, вызовите функцию `ImageArc()`:

```
ImageArc($image, $x, $y, $width, $height, $start, $end, $color);
```

Эллипс можно нарисовать, вызвав функцию `ImageArc()` и установив переменную `$start` в `0`, а переменную `$end` – в `360`:

```
ImageArc($image, $x, $y, $width, $height, 0, 360, $color);
```

Если вызвать функцию `ImageArc()`, установив переменную `$start` в `0`, переменную `$end` – в `360` и задав одинаковые значения для переменных `$width` и `$height`, то получится окружность:

```
ImageArc($image, $x, $y, $diameter, $diameter, 0, 360, $color);
```

Обсуждение

Функция `ImageArc()` отличается чрезвычайной гибкостью, – передавая ей нужные значения, можно без труда создавать простые кривые, такие как эллипсы и окружности. Как и для многих других функций библиотеки GD, первым параметром является область отображения. Следующие два параметра – это координаты x и y центра дуги. За ними – ширина и высота дуги. Поскольку окружность – это дуга с шириной, равной высоте, то для рисования окружности установите значения обоих параметров, соответствующие диаметру окружности.

Шестой и седьмой параметры представляют начальный и конечный углы в градусах. Значение, равное 0, означает 3 часа. Затем дуга идет по часовой стрелке, так что 90° означают 6 часов, 180° означают 9 часов, а 270° соответствуют 12 часам. (Будьте осторожны! Такое поведение свойственно не всем функциям библиотеки GD. Например, при вращении текста используется направление, обратное вращению часо-

вой стрелки.) Центр дуги находится в точке ($x,$y), поэтому если вы рисуете полуокружность от 0° до 180°, ее начало будет располагаться не в точке ($x,$y), а в точке ($x+($diameter/2),$y).

Как обычно, последний параметр – это цвет дуги.

Например, следующий фрагмент рисует в центре области отображения незакрашенную черную окружность диаметром в 100 пикселов, которая изображена в левой половине рис. 15.3:

```
$image = ImageCreate(100,100);
$bg = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageArc($image, 50, 50, 100, 100, 0, 360, $black);
```

Для получения закрашенного эллипса или окружности вызывайте функцию ImageFillToBorder():

```
ImageArc($image, $x, $y, $diameter, $diameter, 0, 360, $color);
ImageFillToBorder($image, $x, $y, $color, $color);
```

Функция ImageFillToBorder() заливает область с началом в точке ($x,$y) цветом, указанным в последнем параметре, вплоть до границы области отображения, или до встречи с линией, окрашенной цветом, заданным третьим параметром.

Вставив это в предыдущие примеры, получаем:

```
$image = ImageCreate(100,100);
$bg = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageArc($image, 50, 50, 100, 100, 0, 360, $black);
ImageFillToBorder($image, 50, 50, $black, $black);
```

Вывод показан в правой половине рис. 15.3.

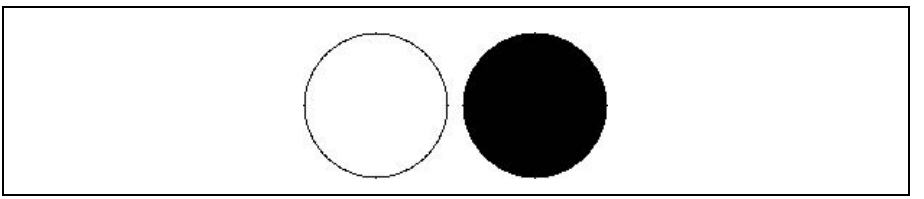

Рис. 15.3. *Черная окружность и закрашенный черным цветом круг*

Пользователи версии GD 2.x могут вызвать функцию ImageFilledArc() и передать ей последний параметр, описывающий стиль заполнения. GD 2.x поддерживает также специальные функции ImageEllipse() и ImageFilledEllipse().

См. также

Дополнительную информацию о рисовании других типов фигур в рецепте 15.1; рецепт 15.3, в котором более подробно описывается рисование с применением стилей и кистей; документацию по функции

ImageArc() на *http://www.php.net/imagearc*, по функции ImageFilledArc() на *http://www.php.net/imagefilledarc* и по функции ImageFillToBorder() на *http://www.php.net/imagefilltoborder*.

15.3. Рисование узорными линиями

Необходимо нарисовать фигуру, используя стиль линии, отличный от стиля по умолчанию, которым является непрерывная линия.

Решение

Для вычерчивания фигур узорными линиями предназначена функция ImageSetStyle(), которой в качестве изображения передается параметр IMG_COLOR_STYLED:

```
$black = ImageColorAllocate($image,   0,   0,   0);
$white = ImageColorAllocate($image, 255, 255, 255);

// создаем черно-белую пунктирную линию толщиной в два пиксела
$style = array($black, $black, $white, $white);
ImageSetStyle($image, $style);

ImageLine($image, 0, 0, 50, 50, IMG_COLOR_STYLED);
ImageFilledRectangle($image, 50, 50, 100, 100, IMG_COLOR_STYLED);
```

Обсуждение

Узор линии (шаблон) определяется массивом цветов. Каждый элемент массива представляет отдельный пиксел кисти. Часто бывает удобно повторять одинаковый цвет последовательных элементов, увеличивая размер черточек в шаблоне.

Например, ниже приведен фрагмент кода, рисующий квадрат с чередующимися белыми и черными пикселами, как показано в левой части рис. 15.4:

```
$style = array($white, $black);
ImageSetStyle($image, $style);
ImageFilledRectangle($image, 0, 0, 49, 49, IMG_COLOR_STYLED);
```

Это тот же самый квадрат, но нарисованный стилем, состоящим из пяти белых пикселов, следующих за пятью черными, как показано в центре рис. 15.4:

```
$style = array($white, $white, $white, $white, $white,
               $black, $black, $black, $black, $black);
```

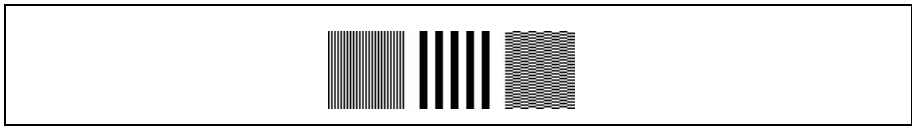

Рис. 15.4. *Три квадрата с чередующимися белыми и черными пикселами*

```
ImageSetStyle($image, $style);
ImageFilledRectangle($image, 0, 0, 49, 49, IMG_COLOR_STYLED);
```

Шаблоны выглядят совершенно разными, хотя оба стиля используют только белые и черные пикселы.

Если кисть не укладывается в фигуру целое число раз, то она продолжает рисовать по кругу. В предыдущем примере представлен квадрат шириной в 50 пикселов. Кисть имеет длину, равную двум пикселам, поэтому укладывается ровно 25 раз; вторая кисть имеет длину 10 пикселов, поэтому укладывается ровно 5 раз. Но если создать квадрат 45×45 и использовать вторую кисть, то прямые линии, как раньше, не получатся (см. правую часть рис. 15.4):

```
ImageFilledRectangle($image, 0, 0, 44, 44, IMG_COLOR_STYLED);
```

См. также

Дополнительную информацию о рисовании фигур в рецептах 15.1 и 15.2; документацию по функции `ImageSetStyle()` на *http://www.php.net/imagesetstyle*.

15.4. Рисование текста

Задача

Вывести текст как графический элемент. Это позволяет динамически отображать кнопки или счетчики выполнения.

Решение

Для встроенных шрифтов библиотеки GD применяется функция `ImageString()`:

```
ImageString($image, 1, $x, $y, 'I love PHP Cookbook', $text_color);
```

Для шрифтов TrueType предназначена функция `ImageTTFText()`:

```
ImageTTFText($image, $size, 0, $x, $y, $text_color, '/path/to/font.ttf',
             'I love PHP Cookbook');
```

Для шрифтов PostScript Type 1 применяются функции `ImagePSLoadFont()` и `ImagePSText()`:

```
$font = ImagePSLoadFont('/path/to/font.pfb');
ImageString($image, 'I love PHP Cookbook', $font, $size,
            $text_color, $background_color, $x, $y);
```

Обсуждение

Для того чтобы поместить текст в область изображения, вызовите функцию `ImageString()`. Подобно другим функциям рисования библиотеки GD, функции `ImageString()` необходима обширная входная инфор-

мация: область, на которой рисуют, номер шрифта, координаты x и y верхнего правого угла первой буквы, отображаемая строка и, наконец, цвет, задаваемый для рисования строки.

В случае функции `ImageString()` имеется пять возможных вариантов выбора шрифта, от 1 до 5. Шрифт номер 1 самый маленький, а шрифт номер 5 самый большой, как показано на рис. 15.5. Любое значение, выходящее за этот диапазон, порождает ближайший к указанному значению допустимый размер.

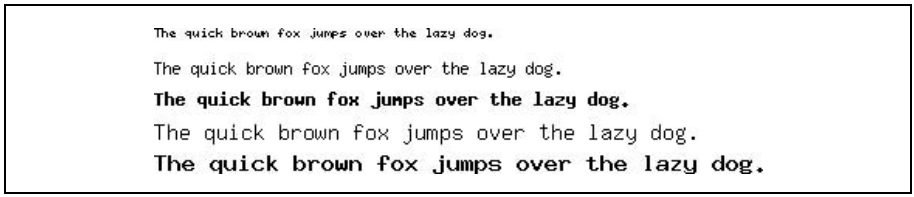

Рис. 15.5. *Размеры встроенных в GD шрифтов*

Чтобы нарисовать текст вертикально, а не горизонтально, используйте вместо предыдущей функцию `ImageStringUp()`. На рис. 15.6 показан ее вывод.

```
ImageStringUp($image, 1, $x, $y, 'I love PHP Cookbook', $text_color);
```

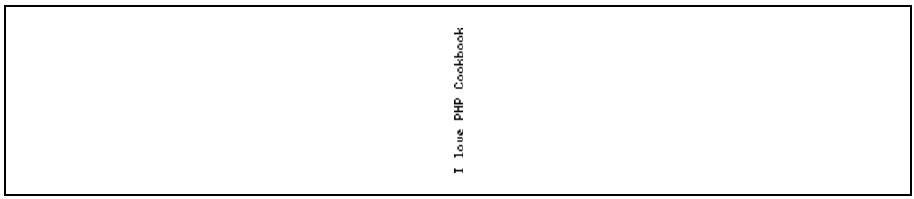

Рис. 15.6. *Вертикальный текст*

Для того чтобы со шрифтами TrueType можно было работать, необходимо также инсталлировать библиотеку FreeType и сконфигурировать PHP во время его инсталляции для использования библиотеки FreeType. Главный сайт FreeType находится на *http://www.freetype.org*. Чтобы разрешить поддержку FreeType 1.x, укажите параметр `--with-ttf`, а для FreeType 2.x – параметр `--with-freetype-dir=DIR`.

Подобно функции `ImageString()`, функция `ImageTTFText()` печатает строку в области отображения, но она имеет немного другие параметры, расположенные в другом порядке:

```
ImageTTFText($image, $size, $angle, $x, $y, $text_color, '/path/to/font.ttf',
             $text);
```

Аргумент `$size` представляет размер шрифта в пикселах, `$angle` – угол поворота в градусах в направлении, обратном направлению вращения часовой стрелки, а */path/to/font.ttf* определяет путь файлу шрифта TrueType. В отличие от функции `ImageString()`, в данном случае точка

($x,$y) – это крайние левые координаты базовой линии первой буквы. Базовая линия определяет нижнюю границу, на которой располагается большинство символов. Такие символы, как «g» и «j», продолжаются ниже базовой линии; символы «a» и «z» находятся на базовой линии.

Шрифтам PostScript Type 1 необходима инсталляция библиотеки *t1lib*. Ее можно загрузить с *ftp://sunsite.unc.edu/pub/Linux/libs/graphics/* и встроить в PHP с помощью параметра --with-t1lib.

И здесь тоже синтаксис печати текста похож, но не тот же самый:

```
$font = ImagePSLoadFont('/path/to/font.pfb');
ImagePSText($image, $text, $font, $size, $text_color,
            $background_color, $x, $y);
ImagePSFreeFont($font);
```

Во-первых, имена шрифтов PostScript нельзя прямо передать функции ImagePSText(), они должны быть загружены с помощью функции ImagePSLoadFont(). В случае успеха функция возвращает ресурс шрифта, который может быть использован функцией ImagePSText(). Во-вторых, помимо указания цвета текста, надо передать цвет фона, который будет использован при вычислении сглаживания. Позиционирование точки ($x,$y) аналогично тому, как это делает библиотека TrueType. Наконец, после окончания работы со шрифтом его можно удалить из памяти, вызвав функцию ImagePSFreeFont().

Кроме указанных выше обязательных аргументов функция ImagePSText() принимает также четыре необязательных аргумента в следующем порядке: space, tightness, angle и antialias_steps. Необходимо включать или все четыре аргумента, или ни один из них (т. е. нельзя передать один, два или три из этих аргументов). Первый аргумент управляет размером физического интервала (т. е. пространством, образующимся после нажатия клавиши пробела); второй аргумент определяет плотность расположения букв; третий аргумент – это угол поворота в градусах в направлении, обратном направлению вращения часовой стрелки; последний аргумент представляет собой значение для сглаживания. Это число должно быть равно или 4, или 16. Для получения более качественной картинки, но требующей больше вычислительных затрат, задайте значение 16.

По умолчанию значения space, tightness и angle равны 0. Положительное значение увеличивает расстояние между словами и буквами или поворачивает рисунок в направлении, обратном направлению вращения часовой стрелки. Отрицательное число сжимает слова и буквы или поворачивает рисунок в обратном направлении. Вывод следующего примера показан на рис. 15.7:

```
// нормальное изображение
ImagePSText($image, $text, $font, $size, $black, $white, $x, $y, 0, 0, 0, 4);

// дополнительное пространство между словами
ImagePSText($image, $text, $font, $size, $black, $white, $x, $y + 30,
            100, 0, 0, 4);
```

```
// дополнительное пространство между буквами
ImagePSText($image, $text, $font, $size, $black, $white, $x, $y + 60,
            0, 100, 0, 4);
```

I love PHP Cookbook

I love PHP Cookbook

I love PHP Cookbook

Рис. 15.7. *Увеличено расстояние между словами и буквами*

См. также

Рецепт 15.5 о рисовании центрированного текста; документацию по функции `ImageString()` на *http://www.php.net/imagestring*, по функции `ImageStringUp()` на *http://www.php.net/imagestringup*, по функции `ImageTTFText()` на *http://www.php.net/imagettftext*, по функции `ImagePSText()` на *http://www.php.net/imagepstext* и по функции `ImagePSLoadFont()` на *http://www.php.net/imagepsloadfont*.

15.5. Рисование центрированного текста

Задача

Необходимо нарисовать текст в центре изображения.

Решение

Определите размер изображения и прямоугольника, ограничивающего текст. По этим координатам вычислите соответствующую область для рисования текста.

Для встроенных в GD шрифтов применяется функция `pc_ImageStringCenter()`, приведенная в примере 15.1.

Пример 15.1. pc_ImageStringCenter( )

```
function pc_ImageStringCenter($image, $text, $font) {

    // размеры шрифта
    $width  = array(1 => 5, 6, 7, 8, 9);
    $height = array(1 => 6, 8, 13, 15, 15);

    // определяем размер изображения
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // определяем размер текста
    $xr = $width[$font] * strlen($text);
    $yr = $height[$font];

    // вычисляем среднюю точку
```

```
    $x = intval(($xi - $xr) / 2);
    $y = intval(($yi - $yr) / 2);

    return array($x, $y);
}
```

Например:

```
list($x, $y) = pc_ImageStringCenter($image, $text, $font);
ImageString($image, $font, $x, $y, $text, $fore);
```

Для шрифтов PostScript следует применять функцию `pc_ImagePSCenter()`, показанную в примере 15.2.

Пример 15.2. pc_ImagePSCenter( )

```
function pc_ImagePSCenter($image, $text, $font, $size, $space = 0,
                          $tightness = 0, $angle = 0) {

    // определяем размер изображения
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // определяем размер текста
    list($xl, $yl, $xr, $yr) = ImagePSBBox($text, $font, $size,
                                   $space, $tightness, $angle);

    // вычисляем среднюю точку
    $x = intval(($xi - $xr) / 2);
    $y = intval(($yi + $yr) / 2);

    return array($x, $y);
}
```

Например:

```
list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
ImagePSText($image, $text, $font, $size, $fore, $back, $x, $y);
```

В примере 15.3 показана функция `pc_ImageTTFCenter()`, предназначенная для шрифтов TrueType.

Пример 15.3. pc_ImageTTFCenter( )

```
function pc_ImageTTFCenter($image, $text, $font, $size) {

    // определяем размер изображения
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // определяем размер текста
    $box = ImageTTFBBox($size, $angle, $font, $text);

    $xr = abs(max($box[2], $box[4]));
    $yr = abs(max($box[5], $box[7]));

    // вычисляем среднюю точку
    $x = intval(($xi - $xr) / 2);
    $y = intval(($yi + $yr) / 2);
```

```
    return array($x, $y);
}
```

Например:

```
list($x, $y) = pc_ImageTTFCenter($image, $text, $font, $size);
ImageTTFText($image, $size, $angle, $x, $y, $fore, $font, $text);
```

Обсуждение

Все три функции из раздела «Решение» возвращают координаты x и y рисунка. Конечно, для вычисления этих координат применяются различные методы, в зависимости от типа шрифта, размера и настроек.

Для шрифтов PostScript Type 1 передайте функции `pc_ImagePSCenter()` изображение, размещенное функцией `ImageCreate()` (или одним из ее аналогов), и набор аргументов, определяющий способ отрисовки текста. Первые три параметра обязательны: отображаемый текст, шрифт и размер шрифта. Следующие три параметры могут отсутствовать: интервал шрифта, плотность расположения букв и угол поворота в градусах.

Для определения размера области изображения внутри функции применяются функции `ImageSX()` и `ImageSY()`, возвращающие ширину и высоту изображения. Затем надо вызвать функцию `ImagePSBBox()`. Она возвращает четыре целочисленных значения: координаты x и y крайнего левого нижнего положения текста и координаты x и y крайнего правого верхнего положения текста. Поскольку координаты отсчитываются относительно базовой линии текста, то обычно они не равны нулю. Например, буква «g» продолжается ниже границы остальных букв; поэтому в данном случае значение для нижнего левого положения отрицательное.

Вооружившись этими шестью значениями, можно вычислить правильное местоположение средней точки. Поскольку координаты верхнего левого угла области отображения равны (0,0), а функции `ImagePSBText()` требуется левый нижний угол, то формулы определения переменных `$x` и `$y` не совпадают друг с другом. Для `$x` мы берем разность между размером области отображения и текста. Это дает размер пространства, окружающего текст. Затем мы делим это число пополам, чтобы определить количество пикселов, которые мы должны оставить слева от текста. Для `$y` мы делаем то же самое, но прибавляем переменные `$yi` и `$yr`. Добавляя эти числа, мы можем определить координату дальней стороны прямоугольника, которая необходима в данном случае из-за обратного порядка отсчета координаты y в библиотеке GD.

Мы умышленно игнорируем левые нижние координаты при выполнении вычислений. Поскольку основная часть текста находится выше базовой линии, то добавление опускающихся пикселов в действительности ухудшает код; кажется, что картинка находится не по центру.

Чтобы отцентрировать текст, соберем все вместе:

```
function pc_ImagePSCenter($image, $text, $font, $size, $space = 0,
                          $tightness = 0, $angle = 0) {

    // определяем размер изображения
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // определяем размер текста
    list($xl, $yl, $xr, $yr) = ImagePSBBox($text, $font, $size,
                                   $space, $tightness, $angle);

    // вычисляем среднюю точку
    $x = intval(($xi - $xr) / 2);
    $y = intval(($yi + $yr) / 2);

    return array($x, $y);
}

$image = ImageCreate(500,500);
$text = 'PHP Cookbook Rules!';
$font = ImagePSLoadFont('/path/to/font.pfb');
$size = 20;
$black = ImageColorAllocate($image, 0, 0, 0);
$white = ImageColorAllocate($image, 255, 255, 255);

list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
ImagePSText($image, $text, $font, $size, $white, $black, $x, $y);
ImagePSFreeFont($font);

header('Content-type: image/png');
ImagePng($image);

ImageDestroy($image);
```

К сожалению, этот пример не работает для встроенных в GD шрифтов TrueType. Нет функции, возвращающей размер строки и работающей со встроенными шрифтами, а функция `ImageTTFBBox()` возвращает восемь значений вместо четырех. Однако сделав некоторые изменения, можно сгладить эти различия.

Встроенные шрифты имеют фиксированную ширину, поэтому можно без труда определить размер символа и создать функцию, возвращающую размер текста на основании его длины. Таблица 15.1 не на 100% корректна, но точность ее результатов находится в пределах одного-двух пикселов, что вполне приемлемо в большинстве случаев.

Таблица 15.1. Размеры символов встроенных шрифтов библиотеки GD

Номер шрифта	Ширина	Высота
1	5	6
2	6	8
3	7	13
4	8	15
5	9	15

Внутри функции pc_ImageStringCenter() мы вычисляем длину строки как целое кратное от ее длины; высота – это просто высота одного символа. Обратите внимание, что функция ImageString() считает ее координату у наивысшей координатой части текста, поэтому при вычислении переменной $y надо изменить знак обратно на минус.

Приведем пример, задействующий все пять шрифтов, в котором выполняется горизонтальное центрирование текста:

```
$text = 'The quick brown fox jumps over the lazy dog.';
for ($font = 1, $y = 5; $font <= 5; $font++, $y += 20) {
    list($x, $y) = pc_ImageStringCenter($image, $text, $font);
    ImageString($image, $font, $x, $y, $text, $color);
}
```

Вывод представлен на рис. 15.8.

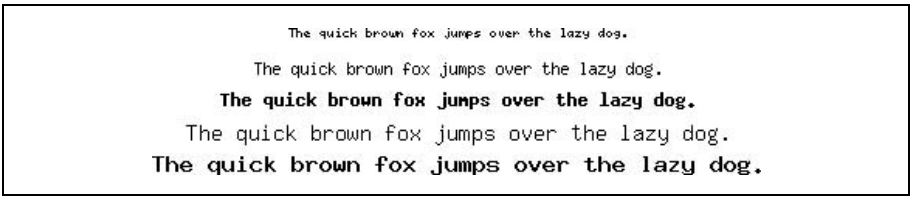

Рис. 15.8. *Центрированные шрифты, встроенные в GD*

Для шрифтов TrueType необходимо использовать функцию ImageTTFBBox() или более современную функцию ImageFtBBox( ) (функция, в имени которой есть буквы «TTF», предназначена для FreeType версии 1.x; функция с буквами «Ft» применяется для FreeType версии 2.x.). Она возвращает восемь чисел: координаты (x,y) четырех углов текста, начиная с нижнего левого и далее по кругу, против часовой стрелки. Таким образом, вторые две координаты – для нижнего правого угла области и т. д.

Чтобы создать функцию pc_ImageTTFCenter(), начните с функции pc_ImagePSCenter() и замените строку:

```
// определяем размер текста
list($xl, $yl, $xr, $yr) = ImagePSBBox($text, $font, $size,
                                $space, $tightness, $angle);
```

на следующий фрагмент:

```
// определяем размер текста
$box = ImageTTFBBox($size, $angle, $font, $text);

$xr = abs(max($box[2], $box[4]));
$yr = abs(max($box[5], $box[7]));
```

Приведем пример применения функции pc_ImageTTFCenter():

```
list($x, $y) = pc_ImageTTFCenter($image, $text, $font, $size);
```

```
ImageTTFText($image, $size, $angle, $x, $y, $white, $black,
             '/path/to/font.ttf', $text);
```

См. также

Дополнительную информацию о рисовании текста в рецепте 15.5; рецепт 15.6, в котором более подробно рассказывается о центрировании текста; документацию по функции `ImageSX()` на *http://www.php.net/imagesx*, по функции `ImageSY()` на *http://www.php.net/imagesy*, по функции `ImagePSBBox()` на *http://www.php.net/imagepsbbox*, по функции `ImageTTFBBox()` на *http://www.php.net/imagettfbbox*, по функции `ImageFtBBox()` на *http://www.php.net/imageftbbox*.

15.6. Построение динамических изображений

Задача

Необходимо создать изображение, основанное на имеющемся шаблоне и динамических данных (как правило, на тексте). Например, надо создать счетчик выполнения.

Решение

Загрузите шаблон изображения, найдите правильное положение, чтобы соответствующим образом отцентрировать текст, добавьте текст к области изображения и пошлите изображение броузеру:

```
// Конфигурационные настройки
$image    = ImageCreateFromPNG('button.png');
$text     = $_GET['text'];
$font     = ImagePSLoadFont('Times');
$size     = 24;
$color    = ImageColorAllocate($image,   0,   0,   0); // черный
$bg_color = ImageColorAllocate($image, 255, 255, 255); // белый

// Печатаем центрированный текст
list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
ImagePSText($image, $text, $font, $size, $color, $bg_color, $x, $y);

// Посылаем изображение
header('Content-type: image/png');
ImagePNG($image);

// Удаляем
ImagePSFreeFont($font);
ImageDestroy($image);
```

Обсуждение

Строить динамические изображения с помощью библиотеки GD довольно легко, достаточно объединить несколько рецептов вместе. В на-

чале программы из раздела «Решение» мы загружаем изображение из запаса шаблонов кнопок; оно играет роль фона, на который накладывается текст. Считаем, что текст поступает прямо из строки запроса. В качестве альтернативы можно получить текст из базы данных (в случае счетчиков доступа) или с удаленного сервера (котировки акций или пиктограммы прогноза погоды).

После этого мы продолжаем работать с другими параметрами: загружаем шрифт и указываем его размер и цвет, а также цвет фона. Однако перед тем как печатать текст, необходимо вычислить его местоположение; функция `pc_ImagePSCenter()` из рецепта 15.6 прекрасно решает эту задачу. Наконец, мы выдаем изображение и удаляем шрифт и зображение из памяти.

Например, следующий фрагмент кода генерирует HTML-страницу и теги изображения с динамическими кнопками, как показано на рис. 15.9:

```
<?php
if (isset($_GET['button'])) {

    // Параметры конфигурации
    $image    = ImageCreateFromPNG('button.png');
    $text     = $_GET['button'];      // динамически сгенерированный текст
    $font     = ImagePSLoadFont('Times');
    $size     = 24;
    $color    = ImageColorAllocate($image,   0,   0,   0); // черный
    $bg_color = ImageColorAllocate($image, 255, 255, 255); // белый

    // Печатаем центрированный текст
    list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
    ImagePSText($image, $text, $font, $size, $color, $bg_color, $x, $y);

    // Посылаем изображение
    header('Content-type: image/png');
    ImagePNG($image);

    // Удаляем
    ImagePSFreeFont($font);
    ImageDestroy($image);

} else {
?>
<html>
<head>
    <title>Sample Button Page</title>
</head>
<body>
    <img src="<?php echo $_SERVER['PHP_SELF']; ?>?button=Previous"
         alt="Previous" width="132" height="46">
    <img src="<?php echo $_SERVER['PHP_SELF']; ?>?button=Next"
         alt="Next"     width="132" height="46">
</body>
```

```
</html>
<?php
}
?>
```

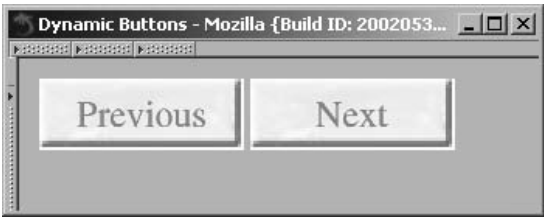

Рис. 15.9. *Простая страница с кнопками*

В этом сценарии, если значение для переменной `$_GET['button']` передано, мы генерируем кнопку и посылаем броузеру PNG. Если значение переменной `$_GET['button']` не установлено, мы печатаем базовую HTML-страницу с двумя внедренными обратными вызовами сценария с запросами для каждой кнопки – один для кнопки Previous (предыдущая) и один для кнопки Next (следующая). Общим решением будет создание отдельной страницы *button.php*, которая возвращает только картинки и устанавливает источник изображения по адресу этой страницы.

См. также

Дополнительную информацию о рисовании текста в рецепте 15.5; рецепт 15.6, в котором более подробно рассказывается о центрировании текста; прекрасное обсуждение кэширования динамических изображений в главе 9 «Graphics» книги «Programming PHP» Кевина Татроэ (Kevin Tatroe) и Расмуса Лердорфа (Rasmus Lerdorf) (O'Reilly).

15.7. Создание и установка прозрачного цвета

Задача

Необходимо сделать один из цветов изображения прозрачным. Фон, на который накладывается изображение, просвечивает через прозрачную часть последнего.

Решение

Это делается с помощью функции `ImageColorTransparent()`:

```
$color = ImageColorAllocate($image, $red, $green, $blue);
ImageColorTransparent($image, $color);
```

Обсуждение

И формат GIF и формат PNG поддерживают прозрачность, а формат JPEG не поддерживает. Для ссылки на прозрачный цвет в библиотеке GD предназначена константа IMG_COLOR_TRANSPARENT. Например, ниже показано, как начертить пунктирную линию, состоящую из чередующихся черных и прозрачных участков:

```
// чертим черно-белую пунктирную линию толщиной в два пиксела
$style = array($black, $black, IMG_COLOR_TRANSPARENT,
IMG_COLOR_TRANSPARENT);
ImageSetStyle($image, $style);
```

Для того чтобы определить текущие установки прозрачности, возьмите значение, возвращенное функцией ImageColorTransparent(), и передайте его функции ImageColorsForIndex():

```
$transparent = ImageColorsForIndex($image, ImageColorTransparent($image));
print_r($transparent);
Array
(
    [red] => 255
    [green] => 255
    [blue] => 255
)
```

Функция ImageColorsForIndex() возвращает массив значений красной, зеленой и синей компонент цвета. В данном случае прозрачным является белый цвет.

См. также

Документацию по функции ImageColorTransparent() на *http://www.php.net/imagecolortransparent* и по функции ImageColorsForIndex() на *http://www.php.net/imagecolorsforindex*.

15.8. Безопасная работа с изображениями

Задача

Необходимо контролировать, кто может просматривать набор изображений.

Решение

Не храните изображения в корневом каталоге документов, а запишите их куда-нибудь в другое место. Чтобы предоставить файл для просмотра, откройте его вручную и пошлите броузеру:

```
header('Content-Type: image/png');
readfile('/path/to/graphic.png');
```

Обсуждение

Первая строка в разделе «Решение» посылает броузеру заголовок `Content-type`, поэтому броузер знает тип прибывающего объекта и отображает его соответствующим образом. Вторая строка открывает файл на диске (или с удаленного URL) для чтения, читает его, выгружает его прямо в броузер и закрывает файл.

Типичный способ работы с изображением заключается в применении тега `<img>` и присвоении атрибуту `src` значения пути к файлу на вашем веб-сайте. Если вы хотите защитить эти изображения, то, возможно, следует использовать какую-либо форму парольной аутентификации. Один из способов – базовая аутентификация HTTP, которой посвящен рецепт 8.9.

Однако типичный способ не всегда самый лучший. Скажем, что произойдет, если необходимо ограничить список файлов, доступных для просмотра, а усложнение работы применением имен пользователей и паролей нежелательно? Один из вариантов состоит в том, чтобы создать ссылки на файлы – если пользователи не могут щелкнуть по ссылке, то они не могут и просмотреть файл. Но пользователи могли сделать закладки на старые файлы или приложить усилия и отгадать другие имена файлов, основываясь на вашей системе именования, и вручную ввести URL в броузере.

Если ваша информация имеет личный характер, то вряд ли вы захотите, чтобы пользователи смогли угадать схему именования и свободно просматривать изображения. Если данные закрытые, то определяется группа лиц, которым дается предварительная информация, для того чтобы они могли писать об этой теме или быть готовыми к распространению этой информации, как только запрет будет снят. Можно это урегулировать, поместив в корневой каталог документов только разрешенную к просмотру информацию, но это потребует значительных перемещений файлов из каталога в каталог. Вместо этого можно хранить файлы в определенном месте и доставлять пользователям только файлы, прошедшие проверку в программе.

Предположим, у вас есть контракт с издательством о повторном показе одного из их комиксов на вашем сайте. Вы не хотите создавать виртуальный архив, поэтому согласны разрешить пользователям просматривать только сценарии последних двух недель. Чтобы увидеть все остальное, они должны посетить официальный сайт. Кроме того, вы можете получать комиксы до даты их официального опубликования, но не желаете пускать их в свободный предварительный просмотр; вы хотите, чтобы пользователи посещали ваш сайт ежедневно.

Вот решение. Поступающие файлы именуются по дате, поэтому легко определить, когда появился определенный файл. Теперь, чтобы заблокировать сценарии, не относящиеся к 14-дневному ролику, используйте следующую программу:

```
// показываем комикс, если он не старше 14 дней и не относится
// к будущим комиксам

// вычисляем текущую дату
list($now_m,$now_d,$now_y) = explode(',',date('m,d,Y'));
$now = mktime(0,0,0,$now_m,$now_d,$now_y);

// двухчасовая граница, чтобы учесть dst
$min_ok = $now - 14*86400 - 7200; // 14 дней назад
$max_ok = $now + 7200;            // сегодня

// определяем временную метку запрашиваемого комикса
$asked_for = mktime(0,0,0,$_REQUEST['mo'],$_REQUEST['dy'],$_REQUEST['yr']);

// сравниваем даты
if (($min_ok > $asked_for) || ($max_ok < $asked_for)) {
    echo 'You are not allowed to view the comic for that day.';
} else {
    header('Content-type: image/png');
    readfile("/www/comics/$_REQUEST['mo']$_REQUEST['dy']
$_REQUEST['yr'].png");
}
```

См. также

Более подробную информацию о чтении файлов в рецепте 18.5.

15.9. Программа: создание гистограмм результатов голосования

Показ результатов голосования может быть более наглядным при использовании красочной гистограммы, вместо простой распечатки результатов в виде текста. Функция, представленная в примере 15.4, использует библиотеку GD для создания изображения, которое показывает совокупные ответы на вопрос, поставленный на голосование.

Пример 15.4. Графические гистограммы

```
Function pc_bar_chart($question, $answers) {

    // определяем цвета для рисования полос
    $colors = array(array(255,102,0), array(0,153,0),
                    array(51,51,204), array(255,0,51),
                    array(255,255,0), array(102,255,255),
                    array(153,0,204));

    $total = array_sum($answers['votes']);

    // определяем значения для промежутков и других магических чисел
    $padding = 5;
    $line_width = 20;
    $scale = $line_width * 7.5;
    $bar_height = 10;

    $x = $y = $padding;
```

```
// размещаем большую палитру для рисования, поскольку мы не знаем
// заранее длину изображения
$image = ImageCreate(150, 500);
$bg_color = ImageColorAllocate($image, 224, 224, 224);
$black = ImageColorAllocate($image, 0, 0, 0);

// печатаем вопрос
$wrapped = explode("\n", wordwrap($question, $line_width));
foreach ($wrapped as $line) {
    ImageString($image, 3, $x, $y , $line, $black);
    $y += 12;
}

$y += $padding;

// печатаем ответы
for ($i = 0; $i < count($answers['answer']); $i++) {

    // формируем процентное соотношение
    $percent = sprintf('%1.1f', 100*$answers['votes'][$i]/$total);
    $bar = sprintf('%d', $scale*$answers['votes'][$i]/$total);

    // выбираем цвет
    $c = $i % count($colors); // обрабатываем случаи с большим
                              количеством полос, чем цветов
    $text_color = ImageColorAllocate($image, $colors[$c][0],
                            $colors[$c][1], $colors[$c][2]);

    // рисуем полосу и числа с процентами
    ImageFilledRectangle($image, $x, $y, $x + $bar,
                         $y + $bar_height, $text_color);
    ImageString($image, 3, $x + $bar + $padding, $y,
                "$percent%", $black);

     $y += 12;

     // печатаем ответ
     $wrapped = explode("\n", wordwrap($answers['answer'][$i],
                        $line_width));
     foreach ($wrapped as $line) {
         ImageString($image, 2, $x, $y, $line, $black);
         $y += 12;
     }

     $y += 7;
 }

 // обрезаем изображение с помощью его копирования
 $chart = ImageCreate(150, $y);
 ImageCopy($chart, $image, 0, 0, 0, 0, 150, $y);

 // доставляем изображение
 header ('Content-type: image/png');
 ImagePNG($chart);

 // удаляем
 ImageDestroy($image);
```

```
        ImageDestroy($chart);
    }
```

Для вызова этой программы создайте массив, содержащий два параллельных массива: `$answers['answer']` и `$answer['votes']`. Элемент `$i` каждого массива хранит текст ответа и общее количество голосов, поданных за ответ `$i`. Примерный вывод показан на рис. 15.10.

```
// Акт II. Сцена II.
$question = 'What a piece of work is man?';

$answers['answer'][] = 'Noble in reason';
$answers['votes'][]  = 29;

$answers['answer'][] = 'Infinite in faculty';
$answers['votes'][]  = 22;

$answers['answer'][] = 'In form, in moving, how express and admirable';
$answers['votes'][]  = 59;

$answers['answer'][] = 'In action how like an angel';
$answers['votes'][]  = 45;

pc_bar_chart($question, $answers);
```

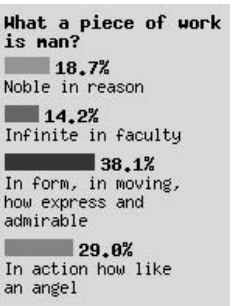

Рис. 15.10. *Графическая гистограмма результатов голосования*

В данном случае ответы назначаются вручную, но в реальном голосовании эта информация может извлекаться из базы данных.

Эта программа представляет собой хорошее начало, но поскольку она использует встроенные в GD шрифты, здесь имеется большое количество внедренных в программу магических чисел, соответствующих высоте и ширине шрифтов. Кроме того, пространственные промежутки между ответами жестко зашиты в программе. Если вы модифицируете программу для использования более продвинутых шрифтов, таких как PostScript или TrueType, то вам придется обновить алгоритм, управляющий этими числами.

В начале функции определяется группа комбинаций RGB; они используются в качестве цветов для рисования полос. Объявлено множество

констант, например: `$line_width` представляет максимальное количество символов в строке, переменная `$bar_height` определяет высоту полос, а `$scale` определяет масштаб длины полосы как функцию от самой длинной строки. Переменная `$padding` нужна для того, чтобы отодвинуть результаты на пять пикселов от границы области отображения.

Затем мы создаем очень большую область отображения для рисования диаграммы; позднее мы обрежем область отображения снизу, но заранее трудно определить, насколько большим будет общий размер. По умолчанию фоновый цвет гистограммы (224, 224, 224), или светло-серый.

Для того чтобы разумно ограничить ширину диаграммы, мы используем функцию `wordwrap()` для разбиения переменной `$question` на строки меньшего размера и ограничиваем их символом `\n` с помощью функции `explode()`. Это дает нам массив строк с правильными размерами, которые мы будем печатать в цикле по одной строке за раз.

Напечатав вопрос, переходим к ответам. Во-первых, форматируем числа результатов с помощью функции `sprintf()`. Чтобы отформатировать общий процент голосов, поданных за определенный ответ, в виде числа с плавающей точкой и одним десятичным разрядом, указываем формат `%1.1f`. Для определения длины полосы, соответствующей этому числу, задаем подобное число, но умножаем его не на 100, а на магическое число `$scale`, а затем возвращаем целое значение.

Цвет текста извлекается из массива `$colors`, содержащего RGB-тройки. Затем мы вызываем функцию `ImageFilledRectangle()` для рисования полосы и функцию `ImageString()` для рисования текста с процентами справа от полосы. После закрашивания мы печатаем ответ, используя тот же алгоритм, что и для печати вопроса.

Когда все ответы напечатаны, общий размер гистограммы сохраняется в переменной `$y`. Теперь мы можем корректно уменьшить размер рисунка, но у нас нет функции `ImageCrop()`. Чтобы выйти из положения, мы создаем новую область соответствующего размера и применяем функцию `ImageCopy()` к той части оригинального изображения, которую мы хотим сохранить. Затем выдаем изображение соответствующего размера в формате PNG, используя функцию `ImagePNG()`, и удаляем все из памяти с помощью двух вызовов функции `ImageDestroy()`.

Как упоминалось в начале этого раздела, это простая, разработанная «на скорую руку» программа для печати гистограмм. Она работает и решает некоторые проблемы, например выполняет перенос строк, но она не абсолютно безупречна. К примеру, она не очень хорошо настраивается. Многие установки зашиты непосредственно в коде. Тем не менее она демонстрирует, как совместно применять множество функций из библиотеки GD для создания полезного графического приложения.

16

Интернационализация и локализация

16.0. Введение

Каждый, кто пишет программы на PHP, должен, в конце концов, изучить английский язык как минимум в объеме, необходимом для работы с именами функций и конструкциями языка, но сам PHP может создавать приложения, разговаривающие практически на любом языке. Некоторые приложения предназначены для пользователей, разговаривающих на нескольких языках. Поддержка интернационализации и локализации в PHP облегчает работу с приложением, написанным для англоговорящих, пользователям, говорящим по-немецки.

Интернационализация (часто ее сокращенно называют I18N[1]) – это процесс, в ходе которого приложение, разработанное только для одного региона, перестраивается таким образом, чтобы с ним можно было работать в различных регионах. Локализация (ее часто сокращенно называют L10N[2]) – это процесс добавления поддержки нового региона в интернациональное приложение.

«Локаль» (местная специфика, locale) – это группа настроек, описывающих особенности форматирования текста и языка в определенном регионе мира. Настройки делятся на шесть категорий:

LC_COLLATE

Эти настройки управляют порядком сортировки текста в алфавитном порядке.

LC_CTYPE

Эти настройки определяют соответствие между буквами верхнего и нижнего регистров, а также состав различных классов символов, например алфавитно-цифровых.

---

[1] В слове «internationalization» 18 букв между первой «i» и последней «n».

[2] В слове «localization» 10 букв между первой «l» и последней «n».

LC_MONETARY

Эти настройки описывают предпочтительный информационный формат денежного обращения, например символ, выступающий в качестве десятичной точки, или способ обозначения отрицательных сумм.

LC_NUMERIC

Эти настройки описывают предпочтительный формат числовой информации, например разбиение цифр, составляющих числа, на группы и символ, отделяющий тысячи.

LC_TIME

Эти настройки описывают предпочтительный формат информации о времени и дате, например о способе именования месяцев и дней и о том, использовать ли 24-часовой формат времени или 12-часовой.

LC_MESSAGES

Эта категория содержит текстовые сообщения для приложений, которым требуется отображать информацию на различных языках.

Есть также метакатегория LC_ALL, которая включает в себя все названные категории.

Как правило, имя локали состоит из трех компонентов. Во-первых, обязательно указывать сокращение, обозначающее язык, например «en» для английского языка или «pt» для португальского языка. Затем после символа подчеркивания следует необязательный спецификатор страны, позволяющий выделить различные страны, в которых разговаривают на разных диалектах одного и того же языка. Например, «en_US» для американского английского и «en_GB» для британского английского или «pt_BR» для бразильского португальского и «pt_PT» для португальского языка, на котором говорят в Португалии. Наконец, после точки идет необязательный спецификатор набора символов. Например, «zh_TW.Big5» для тайваньского китайского языка использует набор символов Big5. Несмотря на то что большинство имен локалей следуют этим соглашениям, для некоторых имен они не выполняются. Одна из сложностей работы с локалями состоит в том, что они могут быть названы произвольно. Поиск и установка локали обсуждается в рецептах с 16.1 по 16.3.

Для правильной локализации простого текста, времени, дат и денежного обращения необходимы различные методы. Локализация также может быть применена к внешним элементам, используемым вашей программой, таким как изображения и включаемые файлы. Локализации такого типа содержимого посвящены рецепты с 16.4 по 16.8.

Системы для работы с большими объемами данных локализации обсуждаются в рецептах 16.9 и 16.10. Рецепт 16.9 демонстрирует некоторые простые способы управления данными, а рецепт 16.10 знакомит с GNU *gettext* – полным набором средств, обеспечивающих поддержку локализации.

В PHP также реализована поддержка Unicode. Прямое и обратное преобразование данных в кодировку Unicode UTF-8 рассматривается в рецепте 16.11.

16.1. Перечень допустимых локалей

Задача

Необходимо определить, какие локали поддерживает ваша система.

Решение

Список допустимых локалей можно получить с помощью программы *locale*; команда *locale -a* напечатает локали, поддерживаемые вашей системой.

Обсуждение

В системах Linux и Solaris можно найти `locale` в */usr/bin/locale*. В Windows локали перечислены в разделе Язык и Стандарты (Regional Options) панели управления (Control Panel).

Все это будет выглядеть по-разному в различных операционных системах. BSD, например, поддерживает локализацию, но не имеет программы *locale* для получения списка локалей. Локали системы BSD чаще всего находятся в */usr/share/locale*, поэтому, просматривая этот каталог, можно получить список используемых локалей.

Хотя локальная система помогает решить множество задач локализации, ее недостаточная стандартизация может и разочаровать. Системы не обязательно имеют одинаковые локали или, наоборот, могут использовать разные имена для идентичных локалей.

См. также

Страницу *locale(1)* программы man.

16.2. Использование определенной локали

Задача

Необходимо указать PHP, что следует использовать настройки определенной локали.

Решение

Вызовите функцию `setlocale()` с соответствующей категорией и локалью. Покажем, как указать локаль `es_US` (испанский язык, США) для всех категорий:

```
setlocale(LC_ALL,'es_US');
```

Ниже показано, как задать способ форматирования времени и даты для локали de_AT (немецкий язык, Австрия):

```
setlocale(LC_TIME,'de_AT');
```

Обсуждение

Чтобы определить текущую локаль, не меняя ее, вызовите функцию setlocale() со значением NULL в качестве локали:

```
print setlocale(LC_ALL,NULL);
en_US
```

Различные системы также поддерживают множество синонимов для распространенных локалей, перечисленных в файле, таком как */usr/share/locale/locale.alias*. Этот файл состоит из ряда строк, в том числе:

```
russian         ru_RU.ISO-8859-5
slovak          sk_SK.ISO-8859-2
slovene         sl_SI.ISO-8859-2
slovenian       sl_SI.ISO-8859-2
spanish         es_ES.ISO-8859-1
swedish         sv_SE.ISO-8859-1
```

Первая колонка каждой строки – это синоним, во второй показаны локаль и набор символов, на которые указывает синоним. В вызовах функции setlocale() можно использовать синоним вместо соответствующей строки, на которую указывает синоним. Например, можно сделать так:

```
setlocale(LC_ALL,'swedish');
```

вместо:

```
setlocale(LC_ALL,'sv_SE.ISO-8859-1');
```

Чтобы изменить локаль в Windows, откройте панель управления. В разделе Язык и Стандарты (Regional Settings) можно выбрать новую локаль и подобрать остальные настройки.

См. также

Рецепт 16.3, где показано, как установить локаль по умолчанию; документацию по функции setlocale() на *http://www.php.net/setlocale*.

16.3. Установка локали по умолчанию

Задача

Необходимо установить локаль, которую смогут использовать все ваши программы на PHP.

Решение

В начале файла, загруженного конфигурационной директивой `auto_prepend_file`, вызовите функцию `setlocale()` для установки требуемой локали:

```
setlocale(LC_ALL,'es_US');
```

Обсуждение

Даже если соответствующая переменная окружения установлена до запуска веб-сервера или двоичного кода PHP, все равно PHP не может изменить свою локаль до тех пор, пока не будет явно вызвана функция `setlocale()`. После установки переменной окружения `LC_ALL`, например в значение `es_US`, PHP продолжает выполнение, используя локаль C по умолчанию.

См. также

Рецепт 16.2, где показано, как использовать определенную локаль; документацию по функции `setlocale()` на *http://www.php.net/setlocale* и по опции `auto_prepend_file` на *http://www.php.net/manual/en/configuration.directives.php#ini.auto-prepend-file*.

16.4. Локализация текстовых сообщений

Задача

Необходимо отображать текстовые сообщения на языке, соответствующем установленной локали.

Решение

Заведите каталог сообщений, состоящий из слов и фраз, из которого будете извлекать подходящие строки, перед тем как выводить их на печать. Ниже показан простой каталог сообщений на американском и британском вариантах английского, описывающий некоторые продукты питания, и функция, извлекающая слова из этого каталога:

```
$messages = array ('en_US' =>
             array(
              'My favorite foods are' => 'My favorite foods are',
              'french fries' => 'french fries',
              'biscuit'      => 'biscuit',
              'candy'        => 'candy',
              'potato chips' => 'potato chips',
              'cookie'       => 'cookie',
              'corn'         => 'corn',
              'eggplant'     => 'eggplant'
             ),
```

```
            'en_GB' =>
              array(
               'My favorite foods are' => 'My favourite foods are',
               'french fries' => 'chips',
               'biscuit'      => 'scone',
               'candy'        => 'sweets',
               'potato chips' => 'crisps',
               'cookie'       => 'biscuit',
               'corn'         => 'maize',
               'eggplant'     => 'aubergine'
              )
             );

function msg($s) {
  global $LANG;
  global $messages;
  if (isset($messages[$LANG][$s])) {
    return $messages[$LANG][$s];
  } else {
    error_log("l10n error: LANG: $lang, message: '$s'");
  }
}
```

Обсуждение

В следующей короткой программе каталог сообщений нужен для распечатки списка продуктов питания:

```
$LANG = 'en_GB';
print msg('My favorite foods are').":\n";
print msg('french fries')."\n";
print msg('potato chips')."\n";
print msg('corn')."\n";
print msg('candy')."\n";
My favourite foods are:
chips
crisps
maize
sweets
```

Чтобы программа печатала не на британском английском, а на американском, просто присвойте переменной `$LANG` значение `en_US`.

Можно объединить функцию `msg()`, извлекающую сообщения, с функцией `sprintf()` и сохранять фразы, в которых требуется заменять значения. Рассмотрим английскую фразу «I am 12 years old». Или в переводе на испанский: «Tengo 12 años». Испанская фраза не может быть построена соединением перевода слов «I am», числа 12 и слов «years old». Эти строки надо сохранить в каталоге сообщений как строки формата в стиле функции `sprintf()`:

```
$messages = array ('en_US' => array('I am X years old.' =>
                                    'I am %d years old.'),
```

```
                    'es_US' => array('I am X years old.' => 'Tengo %d años.')
            );
```

Затем можно передать результаты функции msg() функции sprintf() в качестве строки формата:

```
$LANG = 'es_US';
print sprintf(msg('I am X years old.'),12);
Tengo 12 años.
```

Для фраз, в которых заменяемые значения на другом языке надо вставить в другом порядке, функция sprintf() поддерживает изменение порядка аргументов:

```
$messages = array ('en_US' =>
                    array('I am X years and Y months old.' =>
                          'I am %d years and %d months old.'),
                   'es_US' =>
                    array('I am X years and Y months old.' =>
                          'Tengo %2$d meses y %1$d años.')
            );
```

Для любого языка функция sprintf() вызывается с тем же самым порядком аргументов (т. е первый аргумент – это годы, а второй – месяцы):

```
$LANG = 'es_US';
print sprintf(msg('I am X years and Y months old.'),12,7);
Tengo 7 meses y 12 años.
```

В строке формата значение %2$ указывает функции sprintf() принять второй аргумент, а значение %1$ – первый аргумент.

Эти фразы можно также хранить как возвращаемое функцией значение, вместо того чтобы помещать их в массив в виде строки. Хранение фраз как возвращаемое функцией значение устраняет необходимость в вызове функции sprintf(). Функции, возвращающие предложения, выглядят так:

```
// Английская версия
function i_am_X_years_old($age) {
 return "I am $age years old.";
}

// Испанская версия
function i_am_X_years_old($age) {
 return "Tengo $age años.";
}
```

Если какие-то части сообщения относятся к массиву, а какие-то – к функциям, то удобным контейнером для каталога языковых сообщений является объект. Ниже показано, как могут выглядеть базовый объект и два простых каталога сообщений:

```
class pc_MC_Base {
  var $messages;
```

```
  var $lang;

  function msg($s) {
    if (isset($this->messages[$s])) {
      return $this->messages[$s];
    } else {
      error_log("l10n error: LANG: $this->lang, message: '$s'");
    }
  }

}

class pc_MC_es_US extends pc_MC_Base {

  function pc_MC_es_US() {
    $this->lang = 'es_US';
    $this->messages = array ('chicken' => 'pollo',
                 'cow'     => 'vaca',
                 'horse'   => 'caballo'
                 );
  }

  function i_am_X_years_old($age) {
    return "Tengo $age años";
  }
}

class pc_MC_en_US extends pc_MC_Base {

  function pc_MC_en_US() {
    $this->lang = 'en_US';
    $this->messages = array ('chicken' => 'chicken',
                 'cow'     => 'cow',
                 'horse'   => 'horse'
                 );
  }

  function i_am_X_years_old($age) {
    return "I am $age years old.";
  }
}
```

Каждый объект каталога сообщений наследует класс `pc_MC_Base`, чтобы получить метод `msg()`, а затем определяет свои собственные сообщения (в своем конструкторе) и свои собственные функции, возвращающие фразы. Покажем, как напечатать текст на испанском языке:

```
$MC = new pc_MC_es_US;

print $MC->msg('cow');
print $MC->i_am_X_years_old(15);
```

Чтобы напечатать тот же самый текст на английском языке, необходимо реализовать переменную `$MC` как объект `pc_MC_en_US`, вместо объекта `pc_MC_es_US`. Остальной код остается неизменным.

См. также

Введение в обсуждение наследования объектов; документацию по функции `sprintf()` на *http://www.php.net/sprintf*.

16.5. Локализация дат и времени

Задача

Необходимо отображать даты и значения времени в соответствии с настройками локали.

Решение

Это делается при помощи строки формата `%c` функции `strftime()`:

```
print strftime('%c');
```

Можно также хранить строки формата функции `strftime()` как сообщения в каталоге сообщений:

```
$MC = new pc_MC_es_US;
print strftime($MC->msg('%Y-%m-%d'));
```

Обсуждение

Строка формата `%c` функции `strftime()` задает возвращаемое этой функцией представление даты и времени, предпочтительное для текущей локали. Ниже показан наиболее быстрый способ форматирования строки времени в соответствии с текущими настройками:

```
print strftime('%c');
```

Этот фрагмент кода генерирует множество результатов:

```
Tue Aug 13 18:37:11 2002     // для локали C по умолчанию
mar 13 ago 2002 18:37:11 EDT // для локали es_US
mar 13 aoы 2002 18:37:11 EDT // для локали fr_FR
```

Форматированная строка времени, полученная с помощью `%c`, хотя и соответствует локали, не очень гибкая. Например, если требуется только время, лучше передать другую строку формата функции `strftime()`. Хуже то, что и сами строки задания формата варьируются от локали к локали. В некоторых локалях значения часов отображаются в диапазоне от 1 до 12 с уточнением А.М./Р.М., а в других они должны изменяться от 0 до 23. Чтобы строки времени выводились корректно именно для данной локали, включите нужные элементы в массив `$messages` для каждого формата времени. Ключ определенного формата времени, например `%H:%M`, для всех локалей будет одинаков. А вот значение может изменяться, например `%H:%M` для 24-часовых локалей или `%I:%M %P` для 12-часовых локалей. Затем найдите соответствующую строку формата и передайте ее функции `strftime()`:

```
$MC = new pc_MC_es_US;

print strftime($MC->msg('%H:%M'));
```

Смена локали не изменяет часовой пояс, она меняет только форматирование показываемого результата.

См. также

Обсуждение строк формата, принимаемых функцией strftime(), в рецепте 3.4; рецепт 3.11, посвященный изменению временных зон в программе; документацию по функции strftime() на *http://www.php.net/strftime*.

16.6. Локализация денежных значений

Задача

Необходимо показать денежную сумму в формате, соответствующем локали.

Решение

Для получения соответствующим образом отформатированной строки предназначена функция pc_format_currency(), показанная в примере 16.1. Например:

```
setlocale(LC_ALL,'fr_CA');
print pc_format_currency(-12345678.45);
(12 345 678,45 $)
```

Обсуждение

Функция pc_format_currency(), показанная в примере 16.1, получает информацию о форматировании значений валюты от функции localeconv(), а затем строит корректную строку с помощью функции number_format() и некоторой логики.

Пример 16.1. pc_format_currency

```
function pc_format_currency($amt) {
    // получаем информацию форматирования значений валюты,
    // соответствующую локали
    $a = localeconv();

    // вычисляем знак переменной $amt, а затем удаляем ее
    if ($amt < 0) { $sign = -1; } else { $sign = 1; }
    $amt = abs($amt);
    // форматируем переменную $amt с соответствующей группировкой,
    // десятичной точкой и дробными значениями
    $amt = number_format($amt,$a['frac_digits'],$a['mon_decimal_point'],
                         $a['mon_thousands_sep']);
```

```
// определяем местоположение символа валюты
// и положительного или отрицательного знаков
$currency_symbol = $a['currency_symbol'];
// is $amt >= 0 ?
if (1 == $sign) {
    $sign_symbol  = 'positive_sign';
    $cs_precedes  = 'p_cs_precedes';
    $sign_posn    = 'p_sign_posn';
    $sep_by_space = 'p_sep_by_space';
} else {
    $sign_symbol  = 'negative_sign';
    $cs_precedes  = 'n_cs_precedes';
    $sign_posn    = 'n_sign_posn';
    $sep_by_space = 'n_sep_by_space';
}
if ($a[$cs_precedes]) {
    if (3 == $a[$sign_posn]) {
        $currency_symbol = $a[$sign_symbol].$currency_symbol;
    } elseif (4 == $a[$sign_posn]) {
        $currency_symbol .= $a[$sign_symbol];
    }
    // символ валюты в начале
    if ($a[$sep_by_space]) {
        $amt = $currency_symbol.' '.$amt;
    } else {
        $amt = $currency_symbol.$amt;
    }
} else {
    // символ валюты после суммы
    if ($a[$sep_by_space]) {
        $amt .= ' '.$currency_symbol;
    } else {
        $amt .= $currency_symbol;
    }
}
if (0 == $a[$sign_posn]) {
    $amt = "($amt)";
} elseif (1 == $a[$sign_posn]) {
    $amt = $a[$sign_symbol].$amt;
} elseif (2 == $a[$sign_posn]) {
    $amt .= $a[$sign_symbol];
}
return $amt;
```

Код в функции `pc_format_currency()`, который ставит символ валюты и знак на правильное место, почти одинаков для положительных и отрицательных сумм; он лишь использует различные элементы массива, возвращенного функцией `localeconv()`. Соответствующие элементы массива, возвращаемого функцией `localeconv()`, показаны в табл. 16.1.

Таблица 16.1. Информация, относящаяся к валюте, возвращаемая функцией localeconv( )

Элемент массива	Описание
currency_symbol	Символ местной валюты
mon_decimal_point	Символ десятичной точки валютной суммы
mon_thousands_sep	Разделитель тысячного разряда валютной суммы
positive_sign	Знак для положительных значений
negative_sign	Знак для отрицательных значений
frac_digits	Количество цифр после запятой
p_cs_precedes	1, если currency_symbol должен предшествовать положительному значению; 0, если должен следовать за ним
p_sep_by_space	1, если пробел должен отделять символ валюты от положительного значения; 0, если нет
n_cs_precedes	1, если currency_symbol должен предшествовать отрицательному значению; 0, если должен следовать за ним
n_sep_by_space	1, если пробел должен отделять currency_symbol от отрицательного значения; 0, если нет
p_sign_posn	Позиция положительного знака: 0, если сумма и currency_symbol должны быть заключены в круглые скобки 1, если знак должен предшествовать сумме и currency_symbol 2, если знак должен следовать за суммой currency_symbol 3, если знак должен стоять непосредственно перед суммой и currency_symbol 4, если знак должен следовать непосредственно за суммой и currency_symbol
n_sign_posn	Положение отрицательного знака: те же значения, что и для p_sign_posn

В С-библиотеке есть функция с именем strfmon(), которая делает для валюты то же самое, что функция strftime() делает для дат и времени, однако в PHP она не реализована. Наша функция pc_format_currency() предоставляет большинство из этих возможностей.

См. также

Рецепт 2.9, в котором также обсуждается функция number_format(); документацию по функции localeconv() на *http://www.php.net/localeconv* и по функции number_format() на *http://www.php.net/number-format*.

16.7. Локализация изображений

Задача

Необходимо показать изображения, в которых имеется текст, причем это должен быть текст на языке, соответствующем локали.

Решение

Создайте каталог изображений для каждой локали, которую собираетесь поддерживать, а также глобальный каталог для изображений, не содержащих информации, специфической для локали. Создайте копии каждого изображения со специфической для локали информацией в соответствующем этой локали каталоге. Убедитесь, что имена файлов изображений одинаковы в различных каталогах. Не выводите изображение непосредственно по его URL, а воспользуйтесь функцией-оболочкой, подобной функции `msg()` из рецепта 16.4, которая печатает специфический для локали текст.

Обсуждение

Функция-оболочка `img()` сначала ищет версию изображения для данной локали, а затем глобальную версию. Если ни одна из версий не найдена, то она выводит сообщение в журнал ошибок:

```
$image_base_path = '/usr/local/www/images';
$image_base_url  = '/images';

function img($f) {
    global $LANG;
    global $image_base_path;
    global $image_base_url;

    if (is_readable("$image_base_path/$LANG/$f")) {
        print "$image_base_url/$LANG/$f";
    } elseif (is_readable("$image_base_path/global/$f")) {
        print "$image_base_url/global/$f";
    } else {
        error_log("l10n error: LANG: $lang, image: '$f'");
    }
}
```

Эта функция должна знать и путь к файлу изображения в данной файловой системе (`$image_base_path`), и путь к изображению, отсчитываемый от базового URL вашего сайта (*/images*). Первый путь нужен ей для проверки возможности чтения файла, а второй – для сборки URL изображения.

Имена файлов локализованных изображений должны быть одинаковыми во всех каталогах локализации. Например, изображение, содержащее сообщение «New!» на фоне желтого звездного взрыва, должно быть названо *new.gif* как в каталоге *images/en_US*, так и в каталоге *ima-*

ges/es_US, даже если файл *images/es_US/new.gif* содержит картину желтого звездного взрыва со словом «¡Nuevo!» на его фоне.

Не забудьте, что текст `alt`, указываемый в тегах изображения, также должен быть локализован. Полностью локализованный тег `<img>` выглядит так:

```
printf('<img src="%s" alt="%s">',img('cancel.png'),msg('Cancel'));
```

Если различные локализованные варианты определенного изображения имеют неодинаковые размеры, то в каталоге сообщений надо сохранить также высоту и ширину изображения:

```
printf('<img src="%s" alt="%s" height="%d" width="%d">',
       img('cancel.png'),msg('Cancel'),
       msg('img-cancel-height'),msg('img-cancel-width'));
```

Локализованные сообщения для `img-cancel-height` и `img-cancel-width` — это не текстовые строки, а целые числа, описывающие размеры изображения *cancel.png* для каждой локали.

См. также

Рецепт 16.4, где обсуждаются специфические для локали каталоги сообщений.

16.8. Локализация включаемых файлов

Задача

Необходимо включить в страницу файлы, имеющие локальные особенности.

Решение

Определив соответствующую локаль, измените `include_path`:

```
$base = '/usr/local/php-include';
$LANG = 'en_US';

$include_path = ini_get('include_path');
ini_set('include_path',"$base/$LANG:$base/global:$include_path");
```

Обсуждение

Переменная `$base` содержит имя базового каталога включаемых локализованных файлов. Файлы, не имеющие локальных особенностей, попадают в *глобальный* каталог, указанный в переменной `$base`, а файлы, имеющие такие особенности, размещаются в подкаталогах, названных в соответствии со своими локалями (т. е. *en_US*). Первым в числе путей, ведущих к включенным файлам, указывается каталог, соответствующий локали, а вторым — глобальный каталог; именно в эти два места обращается PHP в поисках включаемого файла. Такой

порядок указания каталогов гарантирует, что нелокализованная информация будет загружена только в том случае, если локализованные данные окажутся недоступными.

Похожий способ применяется в функции `img()` из рецепта 16.7. Однако в данном случае можно воспользоваться преимуществом директивы PHP `include_path`, позволяющей реализовать автоматический поиск каталогов. Чтобы добиться максимальной выгоды, переустановите `include_path` в своей программе как можно раньше, лучше всего в начале файла, загружаемого с помощью `auto_prepend_file` при каждом вызове.

См. также

Документацию по `include_path` на *http://www.php.net/manual/en/configuration.directives.php#ini.include-path* и по `auto_prepend_file` на *http://www.php.net/manual/en/configuration.directives.php#ini.auto-prepend-file*.

16.9. Управление ресурсами локализации

Задача

Необходимо отслеживать различные каталоги сообщений и изображения.

Решение

Есть два приема, позволяющих упростить управление ресурсами локализации. Первый заключается в создании нового объекта языка, например канадского английского, унаследованного от существующего родственного языка, такого как американский английский. Достаточно заменить слова и фразы нового объекта, отличающиеся от слов и фраз оригинального языка.

Второй прием: надо отметить, какие фразы нового языка еще нуждаются в переводе, и поместить в объект нового языка заглушки, которые должны иметь те же самые значения, что и в базовом языке. Обнаружив одинаковые значения в базовом и в новом языках, можно затем сгенерировать список слов и фраз, подлежащих переводу.

Обсуждение

Программа *catalog-compare.php*, показанная в примере 16.2, выводит сообщения, совпадающие в обоих каталогах, а также сообщения, которые отсутствуют в одном каталоге, но присутствуют в другом.

Пример 16.2. catalog-compare.php

```
$base = 'pc_MC_'.$_SERVER['argv'][1];
$other  = 'pc_MC_'.$_SERVER['argv'][2];
```

```
require 'pc_MC_Base.php';
require "$base.php";
require "$other.php";

$base_obj = new $base;
$other_obj = new $other;

/* Проверяем сообщения другого класса, которые совпадают
 * с сообщениями базового класса или они находятся
 * в базовом классе, но отсутствуют в другом классе */
foreach ($base_obj->messages as $k => $v) {
    if (isset($other_obj->messages[$k])) {
        if ($v == $other_obj->messages[$k]) {
            print "SAME: $k\n";
        }
    } else {
        print "MISSING: $k\n";
    }
}

/* Проверяем сообщения другого класса, которые
 * отсутствуют в базовом классе */
foreach ($other_obj->messages as $k => $v) {
    if (! isset($base_obj->messages[$k])) {
        print "MISSING (BASE): $k\n";
    }
}
```

Чтобы воспользоваться этой программой, поместите каждый объект каталога сообщений в файл с тем же именем, что и имя объекта (т. е. класс `pc_MC_en_US` должен находиться в файле с именем *pc_MC_en_US.php*, а класс `pc_MC_es_US` – в файле *pc_MC_es_US.php*). Затем вызовите программу из командной строки с двумя именами локалей в качестве аргументов:

```
% php catalog-compare.php en_US es_US
```

В контексте веб-разработки целесообразным может оказаться использование отдельной локали и каталога сообщений для каждого запроса. Локаль может поступить из броузера (в заголовке `Accept-Language`) или может быть установлена в явном виде сервером (для отображения одного и того же содержания на разных языках могут быть установлены разные виртуальные хосты). Если для выбора каталога сообщений для каждого запроса требуется один и тот же код, то можно реализовать класс каталога сообщений следующим образом:

```
$classname = "pc_MC_$locale.php";

require 'pc_MC_Base.php';
require $classname.'.php';

$MC = new $classname;
```

См. также

Рецепт 16.4, в котором обсуждаются каталоги сообщений; информацию об обнаружении методов и свойств объекта в рецепте 7.10.

16.10. Расширение gettext

Задача

Необходима комплексная система для работы с каталогами сообщений.

Решение

Задействуйте расширение PHP *gettext*, позволяющее применять утилиты GNU *gettext*:

```
bindtextdomain('gnumeric','/usr/share/locale');
textdomain('gnumeric');

$languages = array('en_CA','da_DK','de_AT','fr_FR');
foreach ($languages as $language) {
  setlocale(LC_ALL, $language);
  print gettext(" Unknown formula").".\n";
}
```

Обсуждение

Расширение *gettext* представляет собой набор инструментов, облегчающих генерацию сообщений на разных языках в приложении. Компиляция PHP с параметром `--with-gettext` открывает доступ к функциям, позволяющим извлекать текст из каталогов сообщений в формате *gettext*; кроме того, существует множество внешних инструментов редактирования каталогов сообщений.

В случае применения *gettext* все сообщения делятся на домены, при этом все сообщения из одного домена хранятся в одном файле. Функция `bindtextdomain()` сообщает *gettext*, где искать каталог сообщений для определенного домена. Вызов:

```
bindtextdomain('gnumeric','/usr/share/locale')
```

означает, что каталог сообщений для домена `gnumeric` в локали `en_CA` находится в файле */usr/share/locale/en_CA/LC_MESSAGES/gnumeric.mo*.

Функция `textdomain('gnumeric')` устанавливает в качестве домена по умолчанию домен `gnumeric`. Вызов `gettext()` извлекает сообщение из домена по умолчанию. Существуют и другие функции, например `dgettext()`, позволяющие извлекать сообщения из различных доменов. Когда вызывается функция `gettext()` (или `dgettext()`), она возвращает требуемое сообщение для текущей локали. Если в каталоге текущей

локали нет сообщения, которое соответствовало бы переданному ей аргументу, то функция `gettext()` (или `dgettext()`) просто возвращает сам аргумент. В результате программа печатает все почему-либо не переведенные сообщения на английском языке (или на любом языке, который является базовым).

Установка домена по умолчанию с помощью функции `textdomain()` делает каждое последующее извлечение сообщения из домена более кратким, поскольку вместо вызова `dgettext('domain','Good morning')` достаточно сделать вызов `gettext('Good morning')`. Однако если даже строка `gettext('Good morning')` выглядит слишком длинной, то можно воспользоваться преимуществом недокументированного синонима `_()` для функции `gettext()`. Вместо `gettext('Good morning')` можно написать `_('Good morning')`.

На веб-сайте расширения *gettext* можно найти полезную и подробную информацию по управлению потоком данных между программистами и переводчиками и о том, как эффективно использовать расширение *gettext*. Сайт также содержит информацию о других инструментах, позволяющих управлять каталогами сообщений, например о специальном режиме GNU Emacs.

См. также

Документацию по *gettext* на *http://www.php.net/gettext*; о библиотеке *gettext* на *http://www.gnu.org/software/gettext/gettext.html*.

16.11. Чтение и запись символов Unicode

Задача

Необходимо читать символы в кодировке Unicode из файла, базы данных или формы; или требуется записать символы в кодировке Unicode.

Решение

Однобайтовые символы в кодировке ISO-8859-1 преобразуются в код UTF-8 с помощью функции `utf8_encode()`:

```
print utf8_encode('Kurt G□del is swell.');
```

Функция `utf8_decode()` используется для преобразования символов в кодировке UTF-8 в однобайтовые символы в кодировке ISO-8859-1:

```
print utf8_decode("Kurt G\xc3\xb6del is swell.");
```

Обсуждение

Существует 256 ASCII-символов. Символы с кодами от 0 до 127 стандартизованы: управляющие символы, буквы и числа, символы пунк-

туации. Однако существуют различные правила для символов с кодами от 128 до 255. Одна из кодировок, называемая ISO-8859-1, охватывает символы, необходимые для письма на большинстве европейских языков, например ц в Gцdel или ñ в pestaña. Однако многим языкам требуется более 256 символов, а набор символов, охватывающий символы более чем одного языка, требует еще больше символов. В этой ситуации положение спасает Unicode; его UTF-8 кодировка позволяет представить более одного миллиона символов.

Такая повышенная функциональность достигается за счет пространства. Для хранения ASCII-символов достаточно одного байта; символам в кодировке UTF-8 требуется до 4 байт. В табл. 16.2 показано побайтное представление символов в кодировке UTF-8.

Таблица 16.2. Побайтное представление кодировки UTF-8

Диапазон кодов символа	Количество байт	Байт 1	Байт 2	Байт 3	Байт 4
0x00000000 - 0x0000007F	1	0xxxxxxx			
0x00000080 - 0x000007FF	2	110xxxxx	10xxxxxx		
0x00000800 - 0x0000FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
0x00010000 - 0x001FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

В табл. 16.2 разряды `x` представляют биты, в которых записаны фактические данные символа. Наименьший значащий бит – это самый правый бит в самом правом байте. В многобайтных символах количество ведущих единичных бит в самом левом байте соответствует количеству байт в коде символа.

См. также

Документацию по функции `utf8_encode()` на *http://www.php.net/utf8-encode* и по функции `utf8_decode()` на *http://www.php.net/utf8-decode*; дополнительную информацию по Unicode, доступную на домашней странице Unicode Consortium, *http://www.unicode.org*; UTF-8 и Unicode FAQ на *http://www.cl.cam.ac.uk/~mgk25/unicode.html* .

17

Интернет-службы

17.0. Введение

Сначала был протокол HTTP, были протоколы FTP, NNTP, IMAP, POP3 и вся сборная солянка остальных протоколов. Люди быстро приняли веб-броузеры, поскольку броузер представлял собой интегрированную программу, позволявшую им проверять почту, читать новостные группы, пересылать файлы и просматривать документы, не беспокоясь о деталях, сопровождающих процесс передачи информации. Для реализации различных протоколов PHP предоставляет и собственные функции, и функции PEAR. С помощью этих функций можно использовать PHP для создания веб-приложений, выполняющих все виды сетевых задач, таких как поиск доменных имен или рассылка сообщений электронной почты. Несмотря на то что PHP упрощает решение этих задач, важно понимать силу и пределы возможностей каждого протокола.

Рецепты с 17.1 по 17.3 посвящены наиболее популярной возможности из всех – электронной почте. Рецепт 17.1 показывает, как посылать простые сообщения электронной почты. Рецепт 17.2 описывает MIME-кодировку электронной почты, позволяющую отправлять простой текст и сообщения в формате HTML. Протоколы IMAP и POP3, применяемые для чтения из почтовых ящиков, описываются в рецепте 17.3.

В следующих двух рецептах обсуждается чтение групп новостей с помощью протокола NNTP. Группы новостей похожи на списки рассылки, но только не каждый пользователь из списка получает письмо, а всем пользователям предоставляется доступ к серверу новостей, на котором они просматривают только те сообщения, которые их интересуют. Группы новостей позволяют также разделить дискуссии на потоки, поэтому легко отслеживать беседы в архивах. Рецепт 17.4 рассказывает о рассылке сообщений, а рецепт 17.5 – об их приеме.

Рецепт 17.6 посвящен обмену файлами с помощью протокола FTP. FTP, или протокол передачи файлов (file transfer protocol), – это способ

отправки и приема файлов через Интернет. FTP-серверы могут требовать пароль при входе пользователя или разрешать анонимный доступ.

Поиск LDAP-серверов – это тема рецепта 17.7, а в рецепте 17.8 обсуждается, как установить подлинность пользователей на LDAP-сервере. LDAP-серверы используются и в качестве адресных, и как централизованное место хранения пользовательской информации. Они оптимизированы для поиска информации и могут быть сконфигурированы так, чтобы их данные дублировались – для обеспечения высокой надежности и короткого времени ответа.

Глава завершается рецептами по работе в сети. Рецепт 17.9 посвящен DNS-преобразованиям доменного имени в IP-адрес и наоборот. Последний рецепт рассказывает, как с помощью PEAR-модуля Ping проверить, активен ли хост, и можно ли получить к нему доступ.

В других разделах книги также рассматриваются некоторые сетевые протоколы. HTTP подробно обсуждается в главе 11. Ее рецепты рассказывают о способах получения доступа к URL. Протоколам, которые объединяют HTTP и XML, посвящена глава 12. В этой главе, наряду с DOM и XSLT, мы обсуждаем развивающуюся область веб-служб, использующую протоколы XML-RPC и SOAP.

17.1. Отправка почты

Задача

Необходимо послать сообщение по электронной почте. Оно может быть непосредственным ответом на действие пользователя, такое как регистрация на вашем сайте, или периодически повторяющимся событием, например еженедельным информационным бюллетенем.

Решение

Для этого надо обратиться к PEAR-классу `Mail`:

```
require 'Mail.php';

$to = 'adam@example.com';

$headers['From'] = 'webmaster@example.com';
$headers['Subject'] = 'New Version of PHP Released!';

$body = 'Go to http://www.php.net and download it today!';

$message =& Mail::factory('mail');
$message->send($to, $headers, $body);
```

Если нельзя использовать PEAR-класс `Mail`, то примените встроенную в PHP функцию `mail()`:

```
$to = 'adam@example.com';
$subject = 'New Version of PHP Released!';
```

```
$body = 'Go to http://www.php.net and download it today!';

mail($to, $subject, $body);
```

Обсуждение

PEAR-класс `Mail` позволяет посылать почту тремя способами. Применяемый метод указывается при создании почтового объекта с помощью `Mail::factory()`.

- Для того чтобы отправить почту с помощью внешней программы, такой как *sendmail* или *qmail*, надо передать параметр `sendmail`.
- Для того чтобы использовать SMTP-сервер, надо передать параметр `smtp`.
- Если передать параметр `mail`, будет вызвана встроенная функция `mail()`. При этом класс `Mail` возьмет настройки из вашего файла *php.ini*.

Кроме параметров `sendmail` и `smtp` необходимо передать второй параметр, указывающий ваши настройки. Чтобы использовать `sendmail`, укажите `sendmail_path` и `sendmail_args`:

```
$params['sendmail_path'] = '/usr/sbin/sendmail';
$params['sendmail_args'] = '-oi -t';

$message =& Mail::factory('sendmail', $params);
```

Одним из подходящих значений для `sendmail_path` является */usr/lib/sendmail*. К сожалению, *sendmail* не имеет точного местоположения; оно меняется от системы к системе, поэтому при попытке отыскать эту программу могут возникнуть сложности. Если вы не можете ее найти, попробуйте */usr/sbin/sendmail* или обратитесь к вашему системному администратору.

Программе *sendmail* можно передать два полезных флага: `-oi` и `-t`. Флаг `-oi` указывает *sendmail* не рассматривать одиночную точку (`.`) в строке как конец сообщения. Флаг `-t` заставляет программу *sendmail* искать в файле сообщения заголовок `To:` и другие строки заголовков.

Те, кто предпочитает *qmail*, могут попытаться использовать */var/qmail/bin/qmail-inject* или */var/qmail/bin/sendmail*.

Работая в Windows, можно использовать SMTP-сервер, поскольку на Windows-машинах не инсталлированы копии программы *sendmail*. Чтобы выбрать SMTP-сервер, передайте параметр `smtp`:

```
$params['host'] = 'smtp.example.com';

$message =& Mail::factory('smtp', $params);
```

В режиме `smtp` можно передать пять необязательных параметров. Параметр `host` — это имя хоста, на котором работает SMTP-сервер; по умолчанию его значением является `localhost`. Параметр `port` представляет порт соединения; по умолчанию это порт `25`. Чтобы разрешить

SMTP-аутентификацию, установите параметр auth в true. Чтобы позволить серверу опознавать вас, установите параметры username и password. SMTP работает не только на Windows-машинах, но и на серверах под управлением UNIX.

Те, у кого нет PEAR-класса Mail, могут работать со встроенной функцией mail(). Программа, которую функция mail() использует для отправки почты, указывается в конфигурационной переменной sendmail_path в файле *php.ini*. На машине под управлением Windows надо присвоить переменной SMTP имя хоста вашего SMTP-сервера. Адрес, указываемый в поле From, хранится в переменной sendmail_from.

Приведем пример с функцией mail():

```
$to = 'adam@example.com';
$subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';

mail($to, $subject, $body);
```

Первый параметр – это адрес электронной почты получателя, второй представляет тему сообщения, а последний содержит тело сообщения. Можно также указать дополнительные заголовки в необязательном четвертом параметре. Например, ниже показано, как добавить заголовки Reply-To и Organization:

```
$to = 'adam@example.com';
$subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';
$header = "Reply-To: webmaster@example.com\r\n"
         ."Organization: The PHP Group";

mail($to, $subject, $body, $header);
```

Отделите каждый заголовок символами \r\n, но не ставьте символы \r\n в конце последнего заголовка.

Независимо от применяемого метода имеет смысл написать функцию-оболочку, помогающую при отправке почты. Отправка всей почты с помощью этой функции облегчает добавление регистрации и других проверок для каждого отправляемого сообщения:

```
function pc_mail($to, $headers, $body) {
    $message =& Mail::factory('mail');

    $message->send($to, $headers, $body);
    error_log("[MAIL][TO: $to]");
}
```

В этом примере сообщение записывается в журнал ошибок, регистрируя получателя каждого посланного сообщения. Оно предоставляет временную метку, что облегчает отслеживание жалоб, когда кто-нибудь пытается рассылать спам через ваш сервер. Есть и другой вариант – создайте список адресов электронной почты типа «не посылать» (do

not send), что не позволит этим людям когда-либо принять еще одно сообщение с вашего сайта. Можно также проверять правильность синтаксиса всех адресов получателей электронной почты, что уменьшит количество возвращенных сообщений.

См. также

Рецепт 13.5 о регулярных выражениях для проверки подлинности адресов электронной почты; рецепт 17.2 об отправке почтовых сообщений типа MIME; дополнительную информацию о получении почтовых сообщений в рецепте 17.3; документацию по функции `mail()` на *http://www.php.net/mail*; о PEAR-классе `Mail` на *http://pear.php.net/package-info.php?package=Mail*; RFC 822 на *http://www.faqs.org/rfcs/rfc822.html*; две книги Брайена Косталеса (Bryan Costales) и Эрика Оллмэна (Eric Allman) «sendmail, 3rd Edition» (O'Reilly, 2002) и «sendmail Desktop Reference» (O'Reilly, 1997) .

17.2. Отправка почты в кодировке MIME

Задача

Необходимо послать сообщение по электронной почте в кодировке MIME. Например, нужно отправить несколько сообщений, содержащих разделы с простым текстом и в формате HTML, и заставить программу чтения почты, понимающую MIME-кодировку, автоматически отображать соответствующий раздел.

Решение

Это делается с помощью PEAR-класса `Mail_mime`:

```
require 'Mail.php';
require 'Mail/mime.php';

$to = 'adam@example.com, sklar@example.com';

$headers['From'] = 'webmaster@example.com';
$headers['Subject'] = 'New Version of PHP Released!';

// создаем MIME-объект
$mime = new Mail_mime;

// добавляем разделы тела сообщения
$text = 'Text version of email';
$mime->setTXTBody($text);

$html = '<html><body>HTML version of email</body></html>';
$mime->setHTMLBody($html);

$file = '/path/to/file.png';
$mime->addAttachment($file, 'image/png');
```

```
// выбираем MIME-кодированные заголовки и тело сообщения
$headers = $mime->headers($headers);
$body = $mime->get();

$message =& Mail::factory('mail');
$message->send($to, $headers, $body);
```

Обсуждение

PEAR-класс `Mail_mime` предоставляет объектно-ориентированный интерфейс, реализующий внутри себя все элементы, вовлеченные в создание почтового сообщения, которое содержит и текстовые разделы, и разделы в формате HTML. Этот класс напоминает PEAR-класс `Mail`, но в нем тело не определяется как строка текста, а создается объект `Mail_mime` и вызываются его методы, чтобы добавить необходимые разделы к телу сообщения:

```
// создаем объект MIME
$mime = new Mail_mime;

// добавляем разделы тела сообщения
$text = 'Text version of email';
$mime->setTXTBody($text);

$html = '<html><body>HTML version of email</body></html>';
$mime->setHTMLBody($html);

$file = '/path/to/file.txt';
$mime->addAttachment($file, 'text/plain');

// выбираем MIME-кодированные заголовки и тело сообщения
$headers = $mime->headers($headers);
$body = $mime->get();
```

Методы `Mail_mime::setTXTBody()` и `Mail_mime::setHTMLBody()` добавляют разделы тела сообщения в формате простого текста и в формате HTML соответственно. Здесь мы передаем переменные, но можно также передать имя файла объекту `Mail_mime`, чтобы он его прочитал. Для использования этой возможности передайте значение `true` в качестве второго параметра:

```
$text = '/path/to/email.txt';
$mime->setTXTBody($text, true);
```

Чтобы присоединить к сообщению вложение, например графику или архив, вызовите функцию `Mail_mime:: addAttachment()`:

```
$file = '/path/to/file.png';
$mime->addAttachment($file,'image/png');
```

Передайте функции путь к файлу и его MIME-тип.

Составив сообщение, делаем последние приготовления и посылаем его:

```
// выбираем MIME-кодированные заголовки и тело сообщения
$headers = $mime->headers($headers);
$body = $mime->get();

$message =& Mail::factory('mail');
$message->send($to, $headers, $body);
```

Сначала вам надо заставить объект `Mail_mime` обеспечить правильно отформатированные заголовки и тело сообщения. Затем используется родительский класс `Mail` для форматирования сообщения и отправки его с помощью функции `Mail_mime::send()`.

См. также

Рецепт 17.1 об отправке обычной электронной почты; дополнительную информацию о получении электронной почты в рецепте 17.3; о PEAR-классе `Mail_Mime` на *http://pear.php.net/package-info.php?package=Mail_Mime*.

17.3. Чтение почты с помощью IMAP или POP3

Задача

Необходимо прочитать почту с помощью протоколов IMAP или POP3, позволяющих создать почтовый клиент на веб-основе.

Решение

Для решения этой задачи надо обратиться к расширению PHP IMAP, умеющему «разговаривать» и на языке IMAP, и на языке POP3:

```
// открываем IMAP-соединение
$mail = imap_open('{mail.server.com:143}',      'username', 'password');
// или открываем POP3-соединение
$mail = imap_open('{mail.server.com:110/pop3}', 'username', 'password');

// берем список всех почтовых заголовков
$headers = imap_headers($mail);

// берем объект заголовка для последнего сообщения в почтовом ящике
$last = imap_num_msg($mail);
$header = imap_header($mail, $last);

// выбираем тело для того же сообщения
$body = imap_body($mail, $last);

// закрываем соединение
imap_close($mail);
```

Обсуждение

Лежащая в основе библиотека, необходимая PHP для поддержки IMAP и POP3, предлагает, на первый взгляд, бесконечное количество возможностей, позволяющих, по существу, написать законченный почтовый клиент. Однако вместе с этими возможностями мы получаем и бульшую сложность. На самом деле в PHP есть 63 различные функции с именами, начинающимися словом `imap`, при этом не учитывается, что некоторые из них тоже понимают язык POP3 и NNTP.

Однако в своей основе разговор с почтовым сервером прост. Как и в случае применения многих других возможностей PHP, все начинается с открытия соединения и захвата дескриптора:

```
$mail = imap_open('{mail.server.com:143}', 'username', 'password');
```

В данном случае открываем IMAP-соединение с сервером по имени *mail.server.com* на порту 143. Одновременно в качестве второго и третьего аргументов передаются имя пользователя и пароль.

Для того чтобы открыть POP3-соединение, добавьте `/pop3` к концу строки с именем сервера и номером порта. POP3 обычно работает на порту 110, поэтому после имени сервера надо добавить `:110`:

```
$mail = imap_open('{mail.server.com:110/pop3}', 'username', 'password');
```

Чтобы зашифровать ваше соединение с помощью SSL, добавьте `/ssl` в конце точно так же, как вы поступали с `pop3`. Необходимо также убедиться в том, что ваша инсталляция PHP собрана с конфигурационным параметром `--with-imap-ssl`, в дополнение к параметру `--with-imap`. Кроме того, саму системную библиотеку IMAP необходимо собрать с поддержкой SSL. Если вы используете сертификат, подписанный самостоятельно, и хотите предотвратить неудачные попытки проверки его подлинности, добавьте также `/novalidate-cert`. Наконец, большинство SSL-соединений общаются или через порт 993, или через порт 995. Все эти параметры могут располагаться в произвольном порядке, поэтому следующая запись имеет полное право на существование:

```
$mail = imap_open('{mail.server.com:993/novalidate-cert/pop3/ssl}',
                  'username', 'password');
```

Заключение переменной в фигурные скобки внутри строки в двойных кавычках, например `{$var}`, – это способ сообщить PHP, какую точно переменную следует интерполировать. Поэтому, чтобы передать интерполированную переменную в качестве первого параметра функции `imap_open()`, преобразуйте открывающую { в escape-последовательность:

```
$server = 'mail.server.com';
$port = 993;

$mail = imap_open("\{$server:$port}", 'username', 'password');
```

Открыв соединение, можно задавать почтовому серверу вопросы. Список всех сообщений в почтовом ящике можно получить, вызвав функцию imap_headers():

```
$headers = imap_headers($mail);
```

Она возвращает массив, каждый элемент которого представляет собой форматированную строку, соответствующую сообщению:

```
A   189) 5-Aug-2002 Beth Hondl           an invitation (1992 chars)
```

Есть и альтернативный способ извлечения сообщения, который обеспечивают функции imap_header() и imap_body(), позволяющие извлечь заголовок объекта и строку тела:

```
$header = imap_header($message_number);
$body   = imap_body($message_number);
```

Функция imap_header() возвращает объект с несколькими полями. Самые полезные из них – поля subject, fromaddress и udate. Все поля перечислены в табл. 17.2 в рецепте 17.5.

Элемент body – это просто строка, но если сообщение состоит из нескольких частей, например сообщение представлено и в виде простого текста, и в виде HTML-документа, то элемент $body содержит обе версии MIME-строки, описывающие их:

```
------=_Part_1046_3914492.1008372096119
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Plain-Text Message

------=_Part_1046_3914492.1008372096119
Content-Type: text/html
Content-Transfer-Encoding: 7bit

<html>HTML Message</html>
------=_Part_1046_3914492.1008372096119--
```

Чтобы избежать этого, вызывайте функцию imap_fetchstructure() в комбинации с функцией imap_fetchbody(). Это позволит выяснить, как отформатировано тело, и извлечь только требуемую часть:

```
// выбираем текст для сообщения $n
$st = imap_fetchstructure($mail, $n);
if (!empty($st->parts)) {
    for ($i = 0, $j = count($st->parts); $i < $j; $i++) {
        $part = $st->parts[$i];
        if ($part->subtype == 'PLAIN') {
             $body = imap_fetchbody($mail, $n, $i+1);
        }
     }
} else {
    $body = imap_body($mail, $n));
}
```

Если сообщение состоит из нескольких частей, то переменная `$st->parts` содержит массив объектов, которые их описывают. Свойство `part` содержит целочисленное описание MIME-типа основного тела. В табл. 17.1 приведены целые числа и соответствующие им MIME-типы. Свойство `subtype` содержит подтип MIME и показывает, к какому типу относится данная часть – `plain`, `html`, `png` или к какому-нибудь другому, например `octet-stream`.

Таблица 17.1. Значения IMAP MIME-типов

Номер	MIME-тип	Константа PHP	Описание	Примеры
0	`text`	`TYPETEXT`	Неформатированный текст	Простой текст, HTML, XML
1	`multipart`	`TYPEMULTIPART`	Сообщение из нескольких частей	Смешанные данные, данные из формы, подписанные данные
2	`message`	`TYPEMESSAGE`	Инкапсулированное сообщение	Новости, HTTP
3	`application`	`TYPEAPPLICATON`	Данные приложения	Восьмибитовый поток, PDF, Zip
4	`audio`	`TYPEAUDIO`	Музыкальный файл	MP3, RealAudio
5	`image`	`TYPEIMAGE`	Графическое изображение	GIF, JPEG, PNG
6	`video`	`TYPEVIDEO`	Видеоклип	MPEG, Quicktime
7	`other`	`TYPEOTHER`	Все остальное	Модели VRML

См. также

Дополнительную информацию об отправке почты в рецептах 17.1 и 17.3; документацию по функции `imap_open()` на *http: //www.php.net/imap_open*, по функции `imap_header()` на *http://www.php.net/imap-header*, по функции `imap-body()` на *http://www.php.net/imap-body*, и об IMAP в целом на *http://www.php.net/ imap*.

17.4. Отправка сообщений в новостные группы Usenet

Задача

Необходимо отправить сообщение в новостную группу Usenet, например в *comp.lang.php*.

Решение

С помощью функции `imap_mail_compose()` отформатируйте сообщение, а затем передайте сообщение на сервер с помощью сокетов:

```
$headers['from'] = 'adam@example.com';
$headers['subject'] = 'New Version of PHP Released!';
$headers['custom_headers'][] = 'Newsgroups: comp.lang.php';

$body[0]['type'] = TYPETEXT;
$body[0]['subtype'] = 'plain';
$body[0]['contents.data'] = 'Go to http://www.php.net and download it
today!';

$post = imap_mail_compose($headers, $body);

$server = 'nntp.example.com';
$port = 119;

$sh = fsockopen($server, $port) or die ("Can't connect to $server.");
fputs($sh, "POST\r\n");
fputs($sh, $post);
fputs($sh, ".\r\n");
fclose($sh);
```

Обсуждение

В PHP нет встроенной функции, которая может послать сообщение в новостную группу. Поэтому надо открыть прямое сокет-соединение с сервером новостей и послать команды для отправки сообщения. Однако для форматирования почтового отправления и создания заголовков и тела сообщения можно вызвать функцию `imap_mail_compose()`. Каждое сообщение должно иметь три заголовка: адрес отправителя `From:`, тема сообщения `Subject:` и имя новостной группы:

```
$headers['from'] = 'adam@example.com';
$headers['subject'] = 'New Version of PHP Released!';
$headers['custom_headers'][] = 'Newsgroups: comp.lang.php';
```

Создайте массив `$headers` для хранения заголовков сообщения. Можно прямо присвоить значения заголовкам `From:` и `Subject:`, но этого нельзя сделать для заголовка `Newsgroups:`. Поскольку чаще всего функция `imap_mail_compose()` применяется для создания сообщений электронной почты, то заголовок `Newsgroups:` не относится к числу предопределенных. Чтобы обойти это, необходимо добавить его в массив `custom_headers` в качестве элемента.

В массиве `custom_headers` применяется иной синтаксис. Имя заголовка (в нижнем регистре) не выступает в качестве имени элемента, а содержимое заголовка не сохраняется в массиве как значение, вместо этого весь заголовок целиком записывается в массив в качестве значения. Между именем заголовка и его значением надо поставить двоеточие, а затем пробел. Убедитесь в правильности написания `Newsgroups:` с прописной буквы `N`, а заканчивается слово буквой `s`.

Тело сообщения может состоять из нескольких частей. В результате параметр тела, переданный функции imap_mail_compose(), представляет собой массив массивов. В разделе «Решение» была только одна часть, поэтому мы присваиваем значения непосредственно элементу $body[0]:

```
$body[0]['type'] = TYPETEXT;
$body[0]['subtype'] = 'plain';
$body[0]['contents.data'] = 'Go to http://www.php.net and download it
today!';
```

Каждая часть должна иметь MIME-тип и подтип. Данное сообщение написано в кодировке ASCII, поэтому оно имеет тип TYPETEXT и подтип plain. Обратимся снова к табл. 17.1 из рецепта 17.3 за списком констант IMAP MIME-типов и тем, что они представляют. Поле contents.data содержит тело сообщения.

Для преобразования этих массивов в форматированную строку вызовите функцию imap_mail_compose($body, $headers). Она возвращает почтовое сообщение, которое выглядит так:

```
From: adam@example.com
Subject: New Version of PHP Released!
MIME-Version: 1.0
Content-Type: TEXT/plain; CHARSET=US-ASCII
Newsgroups: comp.lang.php

Go to http://www.php.net and download it today!
```

Вооружившись почтовым сообщением для сервера новостей, вызовем функцию fsockopen(), чтобы создать соединение:

```
$server = 'nntp.example.com';
$port = 119;

$sh = fsockopen($server, $port) or die ("Can't connect to $server.");
```

Первый параметр функции fsockopen() — это имя хоста, на котором запущен сервер, а второй параметр представляет используемый порт. Если имя сервера новостей неизвестно, попробуйте имена хостов *news*, *nntp* или *news-server* в вашем домене: например *news.example.com*, *nntp. example.com* или *news-server.example.com*. Если ни одно их них не подходит, то обратитесь к системному администратору. По традиции все серверы новостей работают на порту 119.

Соединившись с сервером, вы посылаете сообщение:

```
fputs($sh, "POST\r\n");
fputs($sh, imap_mail_compose($headers, $body));
fputs($sh, ".\r\n");
```

Первая строка говорит серверу, что вы хотите отправить почтовое сообщение. Вторая строка представляет собственно сообщение. Чтобы обозначить конец сообщения, добавьте строку из одиночного символа точки. В конце каждой строки должны находиться символы возврата

каретки и новой строки. Закройте соединение с помощью вызова `fclose($sh)`.

Каждому сообщению на сервере присваивается уникальное имя, известное как `Message-ID`. Для того чтобы ответить на сообщение, возьмите идентификатор `Message-ID` исходного сообщения и используйте его в качестве значения заголовка `References`:

```
// получено при чтении исходного сообщения
$message_id = '<20030410020818.33915.php@news.example.com>';

$headers['custom_headers'][] = "References: $message_id";
```

См. также

Дополнительную информацию о чтении новостных групп в рецепте 17.5; документацию по функции `imap_mail_compose()` на *http://www.php.net/imap-mail-compose*, по функции `fsockopen()` на *http://www.php.net/fsockopen*, по функции `fputs()` на *http://www.php.net/fputs* и по функции `fclose()` на *http://www.php.net/fclose*; RFC 977 на *http://www.faqs.org/rfcs/rfc977.html*.

17.5. Чтение новостей из Usenet

Задача

Требуется читать новости сети Usenet с помощью протокола NNTP, чтобы разговаривать с сервером новостей.

Решение

Для этого применяется расширение PHP IMAP. Оно также понимает протокол NNTP:

```
// открываем соединение с nntp-сервером
$server = '{news.php.net/nntp:119}';
$group = 'php.general'; // main PHP mailing list
$nntp = imap_open("$server$group", '', '', OP_ANONYMOUS);

// получаем заголовок
$header = imap_header($nntp, $msg);

// извлекаем поля
$subj  = $header->subject;
$from  = $header->from;
$email = $from[0]->mailbox."@".$from[0]->host;
$name  = $from[0]->personal;
$date  = date('m/d/Y h:i A', $header->udate);

// получаем тело
$body  = nl2br(htmlspecialchars(imap_fetchbody($nntp,$msg,1)));

// закрываем соединение
imap_close($nntp);
```

Обсуждение

Для того чтобы читать сообщения с сервера новостей, необходимо установить с ним соединение и указать интересующую группу:

```
// открываем соединение с nntp-сервером
$server = "{news.php.net/nntp:119}";
$group = "php.general";
$nntp = imap_open("$server$group","","",OP_ANONYMOUS);
```

Функция `imap_open()` принимает четыре параметра. Первый определяет сервер новостей и новостную группу, которую требуется прочитать. В данном случае в качестве сервера выступает *news.php.net* – сервер новостей, который является зеркалом всех почтовых списков PHP. Благодаря добавлению `/nntp` расширение IMAP «понимает», что вы читаете новости, а не почту, и определяет номер порта, `119` (типичное значение порта, зарезервированное за протоколом NNTP). NNTP означает транспортный протокол сетевых новостей (Network News Transport Protocol). Он применяется для организации взаимодействия серверов новостей точно так же, как HTTP является протоколом взаимодействия веб-серверов. Группой здесь является *php.general* – основной почтовый список сообщества PHP.

Средние два параметра функции `imap_open()` – это имя пользователя и пароль, на случай, если потребуется провести идентификацию вашей личности. Поскольку сервер *news.php.net* открыт на чтение для всех, оставьте эти параметры пустыми. Наконец, передайте флаг `OP_ANONYMOUS`, который сообщает IMAP, что вы анонимный читатель; сервер не будет делать запись о вас в специальном файле *.newsrc*.

После установления соединения обычно требуется получить или основной список последних сообщений, или все детали одного определенного сообщения. Ниже приведена программа, отображающая недавние сообщения:

```
// читаем и показываем почтовый каталог
$last = imap_num_msg($nntp);
$n = 10; // показываем 10 последних сообщений

// заголовок таблицы
print <<<EOH
<table>
<tr>
    <th align="left">Subject</th>
    <th align="left">Sender</th>
    <th align="left">Date</th>
</tr>
EOH;

// сообщения
for ($i = $last-$n+1; $i <= $last; $i++) {
    $header = imap_header($nntp, $i);
```

```
        if (! $header->Size) { continue; }

        $subj  = $header->subject;
        $from  = $header->from;
        $email = $from[0]->mailbox."@".$from[0]->host;
        $name  = $from[0]->personal ? $from[0]->personal : $email;
        $date  = date('m/d/Y h:i A', $header->udate);

    print <<<EOM
    <tr>
        <td><a href="$_SERVER[PHP_SELF]"?msg=$i\">$subj</a></td>
        <td><a href="mailto:$email">$name</a></td>
        <td>$date</td>
    </tr>
    EOM;
         }

    // нижний колонтитул таблицы
    echo "</table>\n";
```

Для того чтобы просмотреть список почтовых сообщений, необходимо указать нужный номер. Первому сообщению в любой группе присваивается номер 1, а самое свежее сообщение получает номер, возвращенный функцией `imap_num_msg()`. Поэтому чтобы извлечь последние `$n` сообщений, выполните цикл от `$last-$n+1` до `$last`.

Внутри цикла вызовите функцию `imap_header()` для получения информации о почтовом сообщении, содержащейся в заголовке. Заголовок содержит всю метаинформацию, но не фактический текст сообщения; он хранится в теле. Заголовок, как правило, значительно меньше тела, это позволяет быстро получить информацию о многих почтовых сообщениях.

Теперь передайте функции `imap_header()` два параметра: дескриптор соединения с сервером и номер сообщения. Она возвращает объект с множеством свойств, которые перечислены в табл. 17.2.

Таблица 17.2. Поля функции imap_header( ), полученные от NNTP-сервера

Имя	Описание	Тип	Пример
`date` или `Date`	Данные, форматированные в соответствии с RFC 822: `date('r')`	String	Fri, 16 Aug 2002 01:52:24 –0400
`subject` или `Subject`	Тема сообщения	String	Re: PHP Cookbook Revisions
`message_id`	Уникальный идентификатор, определяющий сообщение	String	<20030410020818.33915.php@news.example.com>
`newsgroups`	Имя группы, в которую было послано сообщение	String	php.general

Таблица 17.2 (продолжение)

Имя	Описание	Тип	Пример
toaddress	Адрес, куда было послано сообщение	String	php-general@ lists.php.net
to	Проанализированная версия поля toaddress	Object	mailbox: «php-general», host: «lists-php.net»
fromaddress	Адрес, откуда было послано сообщение	String	Ralph Josephs <ralph@example.net>
from	Проанализированная версия поля fromaddress	Object	personal: «Ralph Josephs», mailbox: «Ralph», host: «example.net»
reply_toaddress	Адрес, по которому нужно ответить, если хотите связаться с автором	String	rjosephs@example.net
reply_to	Проанализированная версия поля reply_ toaddress	Object	Mailbox: «rjosephs», host: «example.net»
senderaddress	Человек, пославший сообщение; обычно всегда совпадает с полем from, но если поле from не определяет однозначно отправителя сообщения, то это делает данное поле	String	Ralph Josephs <ralph@example.net>
sender	Проанализированная версия поля senderaddress	Object	Personal: «Ralph Josephs», mailbox: «Ralph», host: «example.net»
Recent	Если сообщение недавнее или новое с момента последней проверки почты пользователем	String	Y или N
Unseen	Если сообщения не видно	String	Y или « »
Flagged	Если сообщение отмечено	String	Y или « »
Answered	Если на данное сообщение был послан ответ	String	Y или « »
Deleted	Если сообщение удаляется	String	Y или « »
Draft	Если сообщение является черновиком	String	Y или « »
Size	Размер сообщения в байтах	String	1345
udate	UNIX-метка даты/времени сообщения	Int	1013480645
Mesgno	Номер сообщения в группе	String	34943

Вот некоторые самые полезные поля: size, subject, перечень from и udate. Свойство size представляет размер сообщения в байтах; если оно равно нулю, то сообщение было либо удалено, либо убрано каким-либо другим способом. Поле subject показывает тему сообщения. Перечень from – более сложное свойство. Это массив объектов; каждый элемент массива содержит объект с тремя свойствами: personal, mailbox и host. Поле personal хранит имя отправителя: Homer Simpson. Поле mailbox представляет часть адреса электронной почты, находящуюся перед символом «@», – homer. Свойство host – это часть почтового адреса после знака «@» – thesimpsons.com. Обычно в списке from находится только один элемент, поскольку, как правило, сообщение имеет одного отправителя.

Присвоим переменной $from объект $header->from, т. к. PHP не имеет прямого доступа к $header->from[0]->personal из-за массива в середине структуры. Затем объединяем $from[0]->mailbox и $from[0]->host в форме почтового адреса отправителя. Применяем тернарный оператор для присвоения полю personal имени отправителя, если это поле определено, в противном случае берем почтовый адрес.

Поле udate содержит время отправки почты в виде метки даты/времени UNIX. Для преобразования значения поля из секунд в более дружественный для человека формат применяется функция date().

Конкретное почтовое отправление можно просмотреть следующим образом:

```
// читаем и показываем одно сообщение
$header = imap_header($nntp, $msg);

$subj  = $header->subject;
$from  = $header->from;
$email = $from[0]->mailbox."@".$from[0]->host;
$name  = $from[0]->personal;
$date  = date('m/d/Y h:i A', $header->udate);
$body  = nl2br(htmlspecialchars(imap_fetchbody($nntp,$msg,1)));

print <<<EOM
<table>
<tr>
    <th align=left>From:</th>
    <td>$name &lt;<a href="mailto:$email">$email</a>&gt;</td>
</tr>
<tr>
    <th align=left>Subject:</th>
    <td>$subj</td>
</tr>
<tr>
    <th align=left>Date:</th>
    <td>$date</td>
</tr>
<tr>
```

```
    <td colspan="2">$body</td>
</tr>
</table>
EOM;
```

Код для извлечения единственного сообщения подобен коду для получения последовательности заголовков сообщения. Основное отличие состоит в том, что вы определяете переменную `$body`, представляющую собой результат работы цепочки из трех функций. Самая внутренняя функция, `imap_fetchbody()`, вызывается для получения тела сообщения; она принимает те же параметры, что и функция `imap_header()`. Тело сообщения передается функции `htmlspecialchars()`, преобразующей в escape-последовательность любой HTML-элемент, который может пересечься с вашими. Затем ее результат передается функции `nl2br()`, конвертирующей все символы возврата каретки в теги XHTML `<br/>`; теперь сообщение должно выглядеть на веб-странице правильно.

Чтобы прервать соединение с IMAP-сервером и закрыть поток, передайте функции `imap_close()` дескриптор IMAP-соединения:

```
// по окончании закрываем соединение
imap_close($nntp);
```

См. также

Дополнительную информацию об отправке сообщений в новостные группы в рецепте 17.4; документацию по функции `imap_open()` на *http://www.php.net/imap-open*, по функции `imap_header()` на *http://www.php.net/imap-header*, по функции `imap_body()` на *http://www.php.net/imap-body*, по IMAP в целом на *http://www.php.net/imap*; программу на PHP для чтения новостных групп без использования IMAP на *http://cvs.php.net/cvs.php/php-news-web*; RFC 977 на *http://www.faqs.org/rfcs/rfc977.html*.

17.6. Получение и размещение файлов с помощью FTP

Задача

Необходимо переправить файлы с помощью FTP.

Решение

Для этого применяются встроенные FTP-функции PHP:

```
$c = ftp_connect('ftp.example.com')     or die("Can't connect");
ftp_login($c, $username, $password)     or die("Can't login");
ftp_put($c, $remote, $local, FTP_ASCII) or die("Can't transfer");
ftp_close($c);                          or die("Can't close");
```

Можно также использовать расширение cURL:

```
$c = curl_init("ftp://$username:$password@ftp.example.com/$remote");
// переменная $local - это местоположение сохраняемого файла
// на локальной машине
$fh = fopen($local, 'w') or die($php_errormsg);
curl_setopt($c, CURLOPT_FILE, $fh);
curl_exec($c);
curl_close($c);
```

Обсуждение

FTP (File Transfer Protocol) означает протокол передачи файлов и представляет собой способ обмена файлами между двумя компьютерами. В отличие от HTTP-серверов, FTP-сервер легко установить как для отправки файлов, так и для их получения.

Работа с встроенными FTP-функциями не требует дополнительных библиотек, но необходимо предварительно разрешить применение самих этих функций с помощью параметра `--enable-ftp`. Эти функции разработаны специально для FTP, поэтому передача файлов с их помощью организуется без труда.

Все транзакции FTP начинаются с того, что ваш компьютер (локальный клиент), устанавливает соединение с другим компьютером (удаленным сервером):

```
$c = ftp_connect('ftp.example.com')     or die("Can't connect");
```

После того как соединение установлено, необходимо послать на сервер имя пользователя и пароль, тогда удаленный сервер сможет идентифицировать вас и разрешить (или не разрешить) вам войти:

```
ftp_login($c, $username, $password)     or die("Can't login");
```

Некоторые FTP-серверы поддерживают возможность, известную как анонимный доступ FTP. При анонимном доступе FTP пользователь может войти, не имея персональной учетной записи на удаленном сервере. В случае выбора анонимного доступа ваше имя пользователя должно быть определено как `anonymous`, а паролем служит ваш адрес электронной почты.

Ниже показано, как передать файлы с помощью функций `ftp_put()` и `ftp_get()`:

```
ftp_put($c, $remote, $local,  FTP_ASCII) or die("Can't transfer");
ftp_get($c, $local,  $remote, FTP_ASCII) or die("Can't transfer");
```

Функция `ftp_put()` принимает файл, находящийся на вашем компьютере, и копирует его на удаленный сервер; функция `ftp_get()` копирует файл с удаленного сервера на ваш компьютер. В предыдущем фрагменте кода переменная `$remote` хранит путь к удаленному файлу, а переменная `$local` указывает на файл, находящийся на вашем компьютере.

Этим функциям также передаются еще два параметра. Параметр FTP_ASCII (используемый в данном случае) сообщает о том, что файл передается как ASCII-текст. В этом случае завершающий символ перевода строки автоматически конвертируется при переходе от системы к системе. Есть и другой параметр, FTP_BINARY, указываемый для файлов, не являющихся простым текстом; при этом символ перевода строки не конвертируется.

Функции ftp_fget() и ftp_fput() позволяют загрузить и выгрузить файл через уже существующий указатель открытого файла (открытого с помощью функции fopen()) вместо использования точного местоположения в файловой системе. Например, ниже показано, как извлечь и записать файл по существующему указателю файла, $fp:

```
$fp = fopen($file, 'w');
ftp_fget($c, $fp, $remote, FTP_ASCII)   or die("Can't transfer");
```

Наконец, чтобы закрыть соединение с удаленным хостом, вызовите функцию ftp_close():

```
ftp_close($c);                          or die("Can't close");
```

Для того чтобы установить продолжительность соединения в секундах, применяется функция ftp_set_option():

```
// Время до выхода - две минуты:
set_time_limit(120)
$c = ftp_connect('ftp.example.com');
ftp_set_option($c, FTP_TIMEOUT_SEC, 120);
```

Значение по умолчанию равно 90 секундам; однако по умолчанию значение параметра max_execution_time сценария PHP равно 30 секундам. Поэтому надо проверить оба значения, если соединение закрывается слишком рано.

Для того чтобы с расширением cURL можно было работать, необходимо загрузить его с *http://curl.haxx.se/* и установить параметр конфигурации --with-curl во время сборки PHP. Применение cURL начните с создания дескриптора cURL с помощью функции curl_init(), а затем укажите, что вы хотите сделать, вызвав функцию curl_setopt(). Последняя принимает три параметра: источник cURL, имя модифицируемой константы cURL и значение, присваиваемое второму параметру. В разделе «Решение» это константа CURLOPT_FILE:

```
$c = curl_init("ftp://$username:$password@ftp.example.com/$remote");
// переменная $local содержит путь к сохраняемому файлу на локальной машине
$fh = fopen($local, 'w') or die($php_errormsg);
curl_setopt($c, CURLOPT_FILE, $fh);
curl_exec($c);
curl_close($c);
```

Передаем URL в функцию curl_init(). Поскольку URL начинается с ftp://, cURL сам узнает, что должен применяться протокол FTP.

Вместо того чтобы входить на сервер, реализуя отдельный вызов, вставляйте имя пользователя и пароль непосредственно в URL. Затем укажите местоположение сохраняемого файла на вашем сервере. Теперь откройте файл с именем `$local` на запись и передайте дескриптор файла функции `curl_setopt()` в качестве значения константы `CURLOPT_FILE`. Передавая файл, cURL автоматически записывает данные в дескриптор файла. После того как все сконфигурировано, вызывается функция `curl_exec()`, инициализирующая транзакцию, а затем функция `curl_close()`, закрывающая соединение.

См. также

Документацию по расширению FTP на *http://www.php.net/ftp* и по расширению cURL на *http://www.php.net/curl*; RFC 959 на *http://www.faqs.org/rfcs/rfc969.html*.

17.7. Поиск адресов с помощью LDAP

Задача

Необходимо запросить у LDAP-сервера адресную информацию.

Решение

Для этого применяется расширение PHP LDAP:

```
$ds = ldap_connect('ldap.example.com')                 or die($php_errormsg);
ldap_bind($ds)                                         or die($php_errormsg);
$sr = ldap_search($ds, 'o=Example Inc., c=US', 'sn=*') or die($php_errormsg);
$e  = ldap_get_entries($ds, $sr)                       or die($php_errormsg);

for ($i=0; $i < $e['count']; $i++) {
    echo $info[$i]['cn'][0] . ' (' . $info[$i]['mail'][0] . ')<br>';
}

ldap_close($ds)                                        or die($php_errormsg);
```

Обсуждение

LDAP (Lightweight Directory Access Protocol) означает облегченный протокол доступа к справочникам. LDAP-сервер хранит справочную информацию, такую как имена и адреса, и позволяет ее запрашивать. С многих точек зрения это напоминает базу данных, за исключением того, что она оптимизирована для хранения информации о людях.

Кроме того, вместо плоской структуры, предоставляемой базой данных, LDAP-сервер позволяет создать иерархию. Например, сотрудников компании можно разделить по отделам, в которых они работают: на отдел сбыта, технический отдел и производственный отдел; или по региональному признаку: на работающих в Северной Америке, Европе

и Азии. Таким образом, можно без труда найти всех сотрудников определенного подразделения компании.

В случае применения LDAP хранилище адресов называется *источником данных* (*data source*). Каждый элемент хранилища имеет уникальный глобальный идентификатор, известный как *отмеченное имя* (*distinguished name*). Это отмеченное имя содержит как персональное имя, так и информацию о компании. Например, John Q. Smith, работающий в американской компании Example Inc., имеет отмеченное имя `cn=John Q. Smith`, `o=Example Inc.`, `c=US`. В LDAP `cn` означает стандартное имя, `o` – название организации, а `c` – страну.

Необходимо разрешить в PHP поддержку LDAP с помощью параметра `--with-ldap`. Можно загрузить LDAP-сервер с *http://www.openldap.org*. Этот рецепт предполагает наличие базовых знаний LDAP. Дополнительную информацию ищите в статьях из раздела «O'Reilly Network» по адресу *http://www.onlamp.com/topics/apache/ldap*.

Взаимодействие с LDAP-сервером выполняется в четыре этапа: соединение, аутентификация, поиск записей и выход. Помимо поиска можно также добавлять, изменять и удалять записи.

Для того чтобы открыть транзакции, нужно соединиться с определенным LDAP-сервером, а затем идентифицировать себя в рамках процесса, известного как связывание (*binding*):

```
$ds = ldap_connect('ldap.example.com')                or die($php_errormsg);
ldap_bind($ds)                                        or die($php_errormsg);
```

Если передать функции `ldap_bind()` только дескриптор соединения, `$ds`, то связывание будет анонимным. Имя пользователя и пароль, если они необходимы, передаются при связывании в качестве второго и третьего параметра:

```
ldap_bind($ds, $username, $password)                  or die($php_errormsg);
```

После входа можно запросить информацию. Поскольку данные организованы иерархически, то надо указать базовое отмеченное имя в качестве второго параметра. Наконец, передается критерий поиска. Например, ниже показано, как найти всех сотрудников по фамилии `Jones` в компании `Example Inc.`, находящейся в стране `US`:

```
$sr = ldap_search($ds, 'o=Example Inc., c=US', 'sn=Jones')
                  or die($php_errormsg);
$e  = ldap_get_entries($ds, $sr)
                  or die($php_errormsg);
```

После того как функция `ldap_search()` возвратит результат, вызовите функцию `ldap_get_entries()` для получения определенной записи данных. Затем выполните цикл по массиву элементов, `$e`:

```
for ($i=0; $i < $e['count']; $i++) {
    echo $e[$i]['cn'][0] . ' (' . $e[$i]['mail'][0] . ')<br>';
}
```

Не вызывайте функцию `count($e)`, а возьмите вычисленный заранее размер записи, хранящийся в переменной `$e['count']`. Внутри цикла напечатайте первое стандартное имя и адрес электронной почты каждого человека. Например:

```
David Sklar (sklar@example.com)
Adam Trachtenberg (adam@example.com)
```

Функция `ldap_search()` ищет в базе по всему дереву, начиная с отмеченного имени и ниже. Чтобы ограничить результаты поиска определенным уровнем, вызовите функцию `ldap_list()`. Поскольку в этом случае поиск ведется в меньшем множестве записей, то функция `ldap_list()` может оказаться значительно быстрее функции `ldap_search()`.

См. также

Рецепт 17.7 об аутентификации пользователей с помощью LDAP; документацию по LDAP на *http://www.php.net/ldap*; RFC 2251 на *http://www.faqs.org/rfcs/rfc2251.html*.

17.8. Применение LDAP для аутентификации пользователей

Задача

Требуется открыть определенные разделы сайта только для авторизованных пользователей. Информация о пользователях не сравнивается с записями в базе данных, не применяется базовая аутентификация HTTP, а используется LDAP-сервер. Сохранение всей информации о пользователях на LDAP-сервере облегчает их централизованное администрирование.

Решение

Для этого применяется PEAR-класс `Auth`, поддерживающий LDAP-аутентификацию:

```
$options = array('host'     => 'ldap.example.com',
                 'port'     => '389',
                 'base'     => 'o=Example Inc., c=US',
                 'userattr' => 'uid');

$auth = new Auth('LDAP', $options);

// начинаем проверку подлинности
// выводим экран регистрации для анонимных пользователей
$auth->start();

if ($auth->getAuth()) {
    // содержание для авторизованных пользователей
} else {
```

```
    // содержание для анонимных пользователей
}

// выход пользователей
$auth->logout();
```

Обсуждение

LDAP-серверы разработаны для хранения, поиска и извлечения адресов, поэтому они предпочтительнее, чем стандартные базы данных, такие как MySQL или Oracle. LDAP-серверы очень быстрые, позволяют легко реализовать контроль доступа, предоставляя разнообразные права доступа различным группам пользователей; кроме того, сервер могут запрашивать многие другие программы. Например, большинство почтовых клиентов могут использовать LDAP-сервер в качестве адресной книги, поэтому если сообщение отправляется «Джону Смиту», то сервер возвращает адрес электронной почты Джона, *jsmith@example.com*.

PEAR-класс `Auth` позволяет проверять подлинность пользователей путем сравнения их данных в файлах, в базах данных и на LDAP-серверах. Первый параметр представляет тип аутентификации, а второй параметр – это массив с информацией о том, как проверять подлинность пользователя. Например:

```
$options = array('host'     => 'ldap.example.com',
                 'port'     => '389',
                 'base'     => 'o=Example Inc., c=US',
                 'userattr' => 'uid');

$auth = new Auth('LDAP', $options);
```

Этот код создает новый объект `Auth`, проверяющий подлинность по информации LDAP-сервера, находящегося на *ldap.example.com* и работающего через порт `389`. Имя базового каталога `o=Example Inc.`, `c=US`, а имя пользователя сравнивается с атрибутом `uid`. Поле `uid` служит идентификатором пользователя. Обычно это имя пользователя на веб-сайте или регистрационное имя главной учетной записи. Если ваш сервер не хранит атрибуты `uid` для всех пользователей, то можно указать атрибут `cn`. Поле стандартного имени содержит полное имя пользователя, например «John Q. Smith».

Метод `Auth::auth()` принимает также необязательный параметр – имя функции, отображающей регистрационную форму. Эта форма может быть отформатирована в соответствии с вашим желанием; единственное требование, чтобы поля ввода были названы `username` и `password`. Кроме того, форма должна представлять данные с помощью метода POST.

```
$options = array('host'     => 'ldap.example.com',
                 'port'     => '389',
                 'base'     => 'o=Example Inc., c=US',
                 'userattr' => 'uid');
```

```
function pc_auth_ldap_signin() {
    print<<<_HTML_
<form method="post" action="$_SERVER[PHP_SELF]">
Name: <input name="username" type="text"><br />
Password: <input name="password" type="password"><br />
<input type="submit" value="Sign In">
</form>
_HTML_;
}

$auth = new Auth('LDAP', $options, 'pc_auth_ldap_signin');
```

После создания объекта `Auth` пользователь идентифицируется с помощью вызова `Auth::start()`:

```
$auth->start();
```

Если пользователь уже зарегистрирован, то ничего не происходит. Если пользователь анонимный, то выводится форма регистрации. Для проверки подлинности пользователя функция `Auth::start()` соединяется с LDAP-сервером, выполняет анонимное связывание и ищет адрес, для которого пользовательский атрибут, указанный в конструкторе, совпадает с именем пользователя, переданным в форме:

```
$options['userattr'] == $_POST['username']
```

Если метод `Auth::start()` находит только одного человека, удовлетворяющего критерию, то он возвращает отмеченное имя пользователя и пытается выполнить авторизованное связывание, указав отмеченное имя и пароль из формы в качестве регистрационной информации. Затем LDAP-сервер сравнивает пароль с атрибутом `userPassword`, связанным с отмеченным именем. Если они совпадают, то пользователь проходит аутентификацию.

Можно вызвать функцию `Auth::getAuth()`, которая возвращает логическое значение, описывающее статус пользователя:

```
if ($auth->getAuth()) {
    print 'Welcome member! Nice to see you again.';
} else {
    print 'Welcome guest. First time visiting?';
}
```

Класс `Auth` использует встроенный модуль сессии для отслеживания пользователей, поэтому после прохождения аутентификации пользователь остается авторизованным до конца сессии, или его отключают явным образом:

```
$auth->logout();
```

См. также

Рецепт 17.7 о поиске LDAP-серверов; PEAR-класс `Auth` на *http://pear.php.net/package-info.php?package=Auth*.

17.9. Поиск в DNS

Задача

Необходимо определить доменное имя или IP-адрес.

Решение

Для этого предназначены функции `gethostbyname()` и `gethostbyaddr()`:

```
$ip   = gethostbyname('www.example.com'); // 192.0.34.72
$host = gethostbyaddr('192.0.34.72'); // www.example.com
```

Обсуждение

Не всегда можно доверять имени, возвращенному функцией `gethostbyaddr()`. Управляемый DNS-сервер для определенного IP-адреса может, в принципе, вернуть любое имя хоста. Обычно администраторы настраивают свои DNS-серверы на возвращение корректных имен хостов, но злонамеренные пользователи могут сконфигурировать свои DNS-серверы так, чтобы они возвращали неверные имена хостов. Один из способов борьбы с подобным обманом состоит в том, чтобы вызвать функцию `gethostbyname()` для имени хоста, возвращенного функцией `gethostbyaddr()`, и убедиться, что это имя соответствует исходному IP-адресу.

Если функция не может успешно определить IP-адрес или имя хоста, то она возвращает не значение `false`, а переданный ей аргумент. Проверить, успешно ли был определен адрес, можно так:

```
if ($host == ($ip = gethostbyname($host))) {
    // failure
}
```

Значение, возвращенное функцией `gethostbyname()`, присваивается переменной `$ip`, а также проверяется, не равно ли значение `$ip` значению исходной переменной `$host`.

Иногда одно имя хоста может соответствовать нескольким IP-адресам. Чтобы найти все хосты, вызовите функцию `gethostbynamel()`:

```
$hosts = gethostbynamel('www.yahoo.com');
print_r($hosts);
Array
(
    [0] => 64.58.76.176
    [1] => 64.58.76.224
    [2] => 64.58.76.177
    [3] => 64.58.76.227
    [4] => 64.58.76.179
    [5] => 64.58.76.225
    [6] => 64.58.76.178
    [7] => 64.58.76.229
```

```
    [8] => 64.58.76.223
)
```

В отличие от функций gethostbyname() и gethostbyaddr(), функция gethostbynamel() возвращает массив, а не строку.

Можно также решать более сложные задачи, связанные с DNS. Например, с помощью функции getmxrr() можно получить MX-записи:

```
getmxrr('yahoo.com', $hosts, $weight);
for ($i = 0; $i < count($hosts); $i++) {
    echo "$weight[$i] $hosts[$i]\n";
}
5 mx4.mail.yahoo.com
1 mx2.mail.yahoo.com
1 mx1.mail.yahoo.com
```

Для того чтобы выполнить перенос зон, динамические обновления DNS и многое другое, обратитесь к пакету PEAR Net_DNS.

См. также

Документацию по функции gethostbyname() на *http://www.php.net/gethostbyname*, по функции gethostbyaddr() на *http://www.php.net/gethostbyaddr*, по функции gethostbyaddrl() на *http://www.php.net/gethostbyaddrl* и по функции getmxrr() на *http://www.php.net/getmxrr*; PEAR-пакет Net_DNS на *http://pear.php.net/package-info.php?package=Net_DNS*; книгу «DNS and BIND» Пола Альбитца (Paul Albitz) и Крикета Ли (Cricket Liu) (O'Reilly).[1]

17.10. Проверка функционирования хоста

Задача

Необходимо пропинговать хост, чтобы проверить, работает ли он, и можно ли получить к нему доступ.

Решение

Для этого применяется пакет PEAR Net_Ping:

```
require 'Net/Ping.php';

$ping = new Net_Ping;
if ($ping->checkhost('www.oreilly.com')) {
    print 'Reachable';
} else {
    print 'Unreachable';
}

$data = $ping->ping('www.oreilly.com');
```

[1] Альбитц П., Ли К. «DNS и BIND». – Пер. с англ. – СПб: Символ-Плюс, 2002.

Обсуждение

Программа *ping* пытается отправить с вашей машины сообщение другому компьютеру. Если все идет хорошо, то вы получаете ряд данных, описывающих транзакцию. Ошибка означает, что программа *ping* по некоторым причинам не может добраться до хоста.

В случае ошибки функция `Net_Ping::checkhost()` возвращает значение `false`, а функция `Net_Ping::ping()` возвращает константу `PING_HOST_NOT_FOUND`. Если появляется проблема при запуске программы *ping* (поскольку на самом деле `Net_Ping` — всего лишь оболочка этой программы), то возвращается `PING_FAILED`.

Если все в порядке, то вы получаете массив, как показано ниже:

```
$results = $ping->ping('www.oreilly.com');

foreach($results as $result) { print "$result\n"; }
PING www.oreilly.com (209.204.146.22) from 192.168.123.101 :
    32(60) bytes of data.
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=0 ttl=239
    time=96.704 msec
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=1 ttl=239
    time=86.567 msec
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=2 ttl=239
    time=86.563 msec
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=3 ttl=239
    time=136.565 msec
40 bytes from www.oreilly.com (209.204.146.22): icmp_seq=4 ttl=239
    time=86.627 msec

--- www.oreilly.com ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/mdev = 86.563/98.605/136.565/19.381 ms
```

Пакет `Net_Ping` не проводит какого-либо анализа данных, чтобы извлечь отдельные фрагменты информации, такие как доля потерянных пакетов или среднее время прохождения. Но вы можете провести такой анализ самостоятельно:

```
$results = $ping->ping('www.oreilly.com');

// берем последнюю строку массива; эквивалентно неразрушающему
// методу array_pop() или $results[count($results) - 1]
$round_trip = end($results);
preg_match_all('#[ /]([.\d]+)#', $round_trip, $times);

// извлекаем данные
list($min,$avg,$max,$mdev) = $times[1];
// или печатаем
foreach($times[1] as $time) { print "$time\n"; }
83.229
91.230
```

```
103.223
7.485
```

Это регулярное выражение ищет или символ пробела или символ косой черты. Затем оно выбирает последовательность из одного или более чисел и десятичной точки. Чтобы избежать преобразования в escape-последовательность символа /, мы выбираем в качестве разделителя нестандартный символ #.

См. также

Пакет PEAR `Net_Ping` на *http://pear.php.net/package-info.php?package=Net_Ping*.

17.11. Получение информации о доменном имени

Задача

Необходимо найти контактную информацию или подробности о доменном имени.

Решение

Это делается при помощи класса PEAR `Net_Whois`:

```
require 'Net/Whois.php';
$server = 'whois.networksolutions.com';
$query  = 'example.org';
$data = Net_Whois::query($server, $query);
```

Обсуждение

Метод `Net_Whois::query()` возвращает большую текстовую строку, содержимое которой укрепляет нас в мысли о том, как тяжело порой бывает проанализировать различные результаты поиска Whois:

```
Registrant:
Internet Assigned Numbers Authority (EXAMPLE2-DOM)
   4676 Admiralty Way, Suite 330
   Marina del Rey, CA 90292
   US

   Domain Name: EXAMPLE.ORG

   Administrative Contact, Technical Contact, Billing Contact:
      Internet Assigned Numbers Authority  (IANA)  iana@IANA.ORG
      4676 Admiralty Way, Suite 330
      Marina del Rey, CA 90292
      US
      310-823-9358
      Fax- 310-823-8649
```

```
   Record last updated on 07-Jan-2002.
   Record expires on 01-Sep-2009.
   Record created on 31-Aug-1995.
   Database last updated on 6-Apr-2002 02:56:00 EST.

   Domain servers in listed order:

   A.IANA-SERVERS.NET                192.0.34.43
   B.IANA-SERVERS.NET                193.0.0.236
```

Например, для того чтобы выделить имена и IP-адреса серверов доменных имен, сделайте следующее:

```
preg_match_all('/^\s*([\S]+)\s+([\d.]+)\s*$/m', $data, $dns,
               PREG_SET_ORDER);

foreach ($dns as $server) {
    print "$server[1] : $server[2]\n";
}
```

Чтобы получить информацию о домене, необходимо присвоить переменной `$server` корректное имя Whois-сервера, соответствующего этому домену. Если сервер не известен, то запросите *whois.internic.net*:

```
require 'Net/Whois.php';

print Net_Whois::query('whois.internic.net','example.org');
[whois.internic.net]

Whois Server Version 1.3

Domain names in the .com, .net, and .org domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: EXAMPLE.ORG
   Registrar: NETWORK SOLUTIONS, INC.
   Whois Server: whois.networksolutions.com
   Referral URL: http://www.networksolutions.com
   Name Server: A.IANA-SERVERS.NET
   Name Server: B.IANA-SERVERS.NET
   Updated Date: 19-aug-2002


>>> Last update of whois database: Wed, 21 Aug 2002 04:56:56 EDT <<<

The Registry database contains ONLY .COM, .NET, .ORG, .EDU domains and
Registrars.
```

В строке «Whois Server:» говорится, что правильным именем сервера, у которого нужно запросить информацию об *example.org*, является *whois.networksolutions.com*.

См. также

PEAR-класс `Net_Whois` на *http://pear.php.net/package-info.php?package=Net_Whois*.

18

Файлы

18.0. Введение

Как правило, в веб-приложениях ввод/вывод организуется между броузером, сервером и базой данных, но есть множество случаев, когда в этот процесс вовлекаются и файлы. Файловые функции полезны при получении веб-страниц для последующей локальной обработки, хранения информации без применения баз данных и записи информации для обмена с другими приложениями. Постепенно PHP становится инструментом, служащим не только для простого заполнения веб-страниц, и здесь функции файлового ввода/вывода становятся еще более полезными.

Интерфейс файлового ввода/вывода в PHP напоминает интерфейс ввода/вывода в C, хотя он и менее сложен. Основным элементом, идентифицирующим файл, из которого читают или в который записывают, является *дескриптор файла*. Этот дескриптор реализует связь с конкретным файлом и используется для операций с этим файлом. Эта глава в основном посвящена открытию и закрытию файлов, работе с файловым дескриптором в PHP и тому, что можно делать с содержимым файла после его открытия. В главе 19 рассматриваются каталоги и файловые метаданные, такие как права доступа.

Открытие файла */tmp/cookie-data* и сохранение содержимого cookie в этот файл выглядит следующим образом:

```
$fh = fopen('/tmp/cookie-data','w')      or die("can't open file");
if (-1 == fwrite($fh,$_COOKIE['flavor'])) { die("can't write data"); }
fclose($fh)                              or die("can't close file");
```

Функция `fopen()` возвращает дескриптор файла, если ее попытка открыть файл увенчалась успехом. Если она не в состоянии открыть файл (например, из-за неправильно установленных прав доступа), то возвращает значение `false`. Способам открытия файлов посвящены рецепты 18.1 и 18.3.

Функция `fwrite()` записывает значение cookie `flavor` в файл с соответствующим дескриптором. Она возвращает количество записанных байтов. Если она не может записать строку (например, из-за отсутствия прав на запись), то возвращает –1.

Наконец, функция `fclose()` закрывает дескриптор файла. Это делается автоматически по завершении работы сценария, но хорошей практикой является непосредственное закрытие всех открытых файлов. Это предотвращает проблемы, связанные с работой программы при вызове из командной строки, и освобождает системные ресурсы. Это позволяет также проверить код возврата функции `fclose()`. При записи в файл данные буферизуются. И до явного вызова функции `fclose()`данные могут быть реально не записаны на диск. Проверив код возврата, можно обезопасить себя от ошибки переполнения диска при попытке записи.

Так же как и другие процессы, PHP должен обладать соответствующими правами на чтение и запись в файл. Обычно это явно видно при запуске сценариев из командной строки, но может вызвать путаницу, когда сценарий запускается в рамках веб-сервера. Возможно, ваш веб-сервер (и, следовательно, ваши PHP-сценарии) запускаются от имени определенного пользователя, выделенного для веб-службы (или, возможно, от имени пользователя `nobody`). Из соображений безопасности, довольно часто этот пользователь имеет ограниченные права доступа к файлам. Если при выполнении вашего сценария возникают проблемы с файловыми операциями, убедитесь, что пользователь или группа (не ваша лично, а вашего веб-сервера) имеют права на выполнение соответствующих файловых операций. Некоторые установки веб-сервиса позволяют запускать ваш сценарий и от вашего имени, но в этом случае вы должны убедиться, что ваши сценарии не смогут случайно прочитать или перезаписать файлы, которые не являются частью вашего веб-сайта.

Поскольку большинство работающих с файлами функций в случае ошибки просто возвращают значение `false`, придется проделать дополнительную работу, чтобы получить более подробную информацию об этой ошибке. Если конфигурационная директива `track_errors` установлена в `on`, то каждое сообщение об ошибке помещается в глобальную переменную `$php_errormsg`. Включение этой переменной в вывод информации об ошибках облегчает отладку:

```
$fh = fopen('/tmp/cookie-data','w')   or die("can't open: $php_errormsg");
if (-1 == fwrite($fh,$_COOKIE['flavor'])) { die("can't write:
                                                $php_errormsg"); }
fclose($fh)                              or die("can't close: $php_errormsg");
```

Если у вас нет прав на запись в файл */tmp/cookie-data*, то следующий пример завершается сообщением об ошибке:

```
can't open: fopen("/tmp/cookie-data", "w") - Permission denied
```

Существуют некоторые различия в трактовке файлов в операционных системах Windows и UNIX. Чтобы обеспечить правильную работу программы доступа к файлам под UNIX и Windows, вам надо позаботиться о корректной обработке символов-ограничителей строк и путей к файлам.

Ограничитель строки в Windows состоит из двух символов: ASCII 13 (возврат каретки) и следующего за ним ASCII 10 (перевод строки или новая строка). В UNIX же это просто ASCII 10. Имена, данные этим символам в эпоху пишущих машинок, объясняют, почему можно получить текст лесенкой при печати файлов с символами-разделителями системы UNIX. Представьте эти имена как команды валику пишущей машинки или принтеру, который печатает по одному символу за раз. Возврат каретки посылает валик в начало строки, на которой он находится, а перевод строки протягивает бумагу на одну строку вперед. Неправильно сконфигурированный принтер, встречающий файл с символами-разделителями UNIX, покорно следует инструкциям: выполняет перевод строки в конце каждой строчки. При этом происходит переход на следующую строку, но не происходит горизонтального перемещения обратно к левой границе печати. Следующая строка текста начинается с горизонтальным отступом – с того самого места, где закончилась предыдущая.

Функции PHP, в которых символ новой строки выступает в качестве символа-ограничителя строк (например, `fgets()`) работают и в Windows и в UNIX, поскольку символ новой строки присутствует в конце строки на любой из этих платформ.

Символы-ограничители строк удаляются с помощью функции PHP `rtrim()`:

```
$fh = fopen('/tmp/lines-of-data.txt','r') or die($php_errormsg);
while($s = fgets($fh,1024)) {
    $s = rtrim($s);
    // делаем что-нибудь с переменной $s ...
}
fclose($fh)                                or die($php_errormsg);
```

Эта функция удаляет любой завершающий строку пробельный символ, включая ASCII 13 и ASCII 10 (а также символы табуляции и пробела). Если в конце строки находятся символы, которые требуется сохранить, а удалить надо символы возврата каретки и перевода строки, то используйте соответствующее регулярное выражение:

```
$fh = fopen('/tmp/lines-of-data.txt','r') or die($php_errormsg);
while($s = fgets($fh,1024)) {
    $s = preg_replace('/\r?\n$/','',$s);
    // делаем что-нибудь с переменной $s ...
}
fclose($fh)                                or die($php_errormsg);
```

В UNIX и Windows также используются различные символы для разделения каталогов в имени пути. В UNIX это косая черта (/), а в Windows – обратная косая черта (\). Однако PHP легко разбирается в этой ситуации, поскольку версия PHP для Windows также считает символ / разделителем каталогов. Например, следующий код успешно печатает содержимое файла *C:\Alligator\Crocodile Menu.txt*:

```
$fh = fopen('c:/alligator/crocodile menu.txt','r') or die($php_errormsg);
while($s = fgets($fh,1024)) {
    print $s;
}
fclose($fh)                                          or die($php_errormsg);
```

Этот фрагмент кода использует и то обстоятельство, что имена файлов в Windows не чувствительны к регистру. Однако в UNIX все наоборот.

Устранение путаницы с символом конца строки – это не только проблема программы, читающей и записывающей файлы, но и проблема исходного кода самой программы. Если в проекте участвует много разработчиков, то все они должны сконфигурировать свои редакторы так, чтобы в них в качестве конца строки выступал один и тот же символ.

После открытия файла PHP предоставляет множество инструментов для обработки содержащейся в нем информации. В рамках C-подобного интерфейса ввода/вывода PHP есть две основные функции для чтения данных из файла – функция `fread()`, читающая определенное количество байт, и функция `fgets()`, читающая строку за один прием (вплоть до определенного количества байт). Следующая программа обрабатывает строки длиной до 256 байт:

```
$fh = fopen('orders.txt','r') or die($php_errormsg);
while (! feof($fh)) {
    $s = fgets($fh,256);
    process_order($s);
}
fclose($fh) or die($php_errormsg);
```

Учтите, что если файл *orders.txt* будет содержать строку длиной 300 байт, то функция `fgets()` вернет вам только первые 256. Следующий вызов функции `fgets()` вернет еще 44 байта и остановится, когда найдет символ новой строки. И лишь затем следующая функция `fgets()` переместится на новую строку файла. В примерах этой главы функции `fgets()` в большинстве случаев передается второй аргумент, равный 1 048 576 или 1 Мбайт. Это превышает длину строк в большинстве текстовых файлов, но присутствие такого необычного числа должно послужить напоминанием о том, что в случае применения функции `fgets()` надо учитывать возможность появления строки с максимальной длиной.

Многие операции с содержимым файлов, такие как выбор случайной строки (см. рецепт 18.10), концептуально проще (и требуют меньше кода), если весь файл предварительно считывается в строку или в массив. Рецепт 18.5 предлагает метод чтения файла в строку, а функция

`file()` помещает каждую строку файла в массив. Как бы то ни было, конечный выбор должен определяться расходом памяти. Такие приемы могут оказаться особенно губительными, если PHP используется как серверный модуль. В большинстве случаев, когда процесс (например, процесс веб-сервера с включенным в него модулем PHP) занимает память (как это делает PHP при чтении всего файла в строку или массив), он не может отдать эту память операционной системе до тех пор, пока не закончит свою работу. Это означает, что вызов функции `file()` для файла размером 1 Мбайт из PHP, запущенного в качестве модуля Apache, увеличивает размер всего текущего процесса Apache на 1 Мбайт до тех пор, пока соответствующий PHP-процесс не закончится. Если повторить такое действие несколько раз, то эффективность сервера в целом снизится. Можно привести убедительные доводы в пользу одновременной обработки всего файла, но когда вы это делаете, помните о побочных последствиях расходования памяти.

Рецепты с 18.20 по 18.23 посвящены запуску сторонних приложений из PHP-программы. Некоторые операторы выполнения программ или функции предлагают способы запуска программы и чтения ее вывода в один прием (обратные апострофы – backticks), или чтения только последней строки вывода программы (`system()`). PHP может использовать и отдельный канал для запуска программы, передавая ей входную информацию или читая ее вывод. Поскольку канал читает с помощью стандартных функций ввода/вывода (`fgets()` и `fread()`), то вы сами решаете, как реализовать ввод, и можете решать другие задачи в промежутках между чтением порций входных данных. Подобным же образом запись в канал осуществляется с помощью функций `fputs()` и `fwrite()`, поэтому и входные данные можно передавать в программу произвольными приращениями.

Каналам присущи те же самые проблемы доступа, что и обычным файлам. Процесс PHP должен иметь права на выполнение программы, для открытия которой он использует канал. Если у вас возникли затруднения с открытием канала, особенно если PHP запускается от имени специального пользователя веб-сервера, убедитесь, что этот пользователь обладает правом запуска программы, канал к которой вы пытаетесь открыть.

18.1. Создание или открытие локального файла

Задача

Необходимо открыть локальный файл, чтобы читать из него данные, или записывать данные в этот файл.

Решение

Это делается при помощи функции `fopen()`:

```
$fh = fopen('file.txt','r') or die("can't open file.txt: $php_errormsg");
```

Обсуждение

Первый аргумент функции `fopen()` – это имя файла; второй аргумент – режим открытия файла. Режим открытия определяет, какие операции можно выполнять с файлом (см. табл. 18.1) и место, в котором располагается указатель файла после открытия (в начале или в конце файла). В зависимости от режима, файл может сокращаться до нулевой длины после открытия или автоматически создаваться, если указанный в первом аргументе файл не существует.

Таблица 18.1. Файловые режимы функции fopen( )

Режим	Можно читать?	Можно записывать?	Указатель файла	Сокращать?	Создавать?
`r`	Да	Нет	В начале	Нет	Нет
`r+`	Да	Да	В начале	Нет	Нет
`w`	Нет	Да	В начале	Да	Да
`w+`	Да	Да	В начале	Да	Да
`a`	Нет	Да	В конце	Нет	Да
`a+`	Да	Да	В конце	Нет	Да

В не-POSIX системах, таких как Windows, к режиму также необходимо добавлять ключ `b`, если открывается двоичный файл, или чтение и запись строки должны выполняться до встречи с символом NUL (ASCII 0):

```
$fh = fopen('c:/images/logo.gif','rb');
```

Для работы с файлом передайте файловый дескриптор, возвращенный функцией `fopen()`, другой функции ввода/вывода, такой как `fgets()`, `fputs()` и `fclose()`.

Если имя файла, передаваемое функции `fopen()` в первом аргументе, не содержит в себе путь к каталогу, то подразумевается, что файл открывается в каталоге запускаемого сценария (веб-контекст) или в текущем каталоге (контекст командной строки).

Можно специально указать функции `fopen()`, что поиск файла следует осуществлять в каталогах `include_path`, заданных в настройках файла *php.ini*. Для этого надо передать значение 1 в качестве третьего необязательного аргумента. Например, следующий фрагмент ищет файл *file.inc* среди каталогов `include_path`:

```
$fh = fopen('file.inc','r',1) or die("can't open file.inc: $php_errormsg");
```

См. также

Документацию по функции `fopen()` на *http://www.php.net/fopen*.

18.2. Создание временного файла

Задача

Требуется файл для временного хранения данных.

Решение

Вызывайте функцию `tmpfile()`, если файл должен храниться только во время выполнения сценария:

```
$temp_fh = tmpfile();
// записываем некоторые данные во временный файл
fputs($temp_fh,"The current time is ".strftime('%c'));
// файл удаляется по окончании сценария
exit(1);
```

Если же требуется создать произвольный файл, но хранить его надо и после выполнения сценария, то имя файла надо сгенерировать с помощью функции `tempnam()`, а затем вызовите функцию `fopen()`:

```
$tempfilename = tempnam('/tmp','data-');
$temp_fh = fopen($tempfilename,'w') or die($php_errormsg);
fputs($temp_fh,"The current time is ".strftime('%c'));
fclose($temp_fh) or die($php_errormsg);
```

Обсуждение

Функция `tmpfile()` создает файл с уникальным именем и возвращает его дескриптор. Файл удаляется, когда для данного файлового дескриптора вызывается функция `fclose()` или заканчивается выполнение сценария.

Во втором случае имя файла генерирует функция `tempnam()`. Она принимает два аргумента: первый это каталог, а второй – префикс имени файла. Если каталог не существует или в него нельзя записывать, то функция `tempnam()` использует временный системный каталог, заданный переменной окружения `TMPDIR` в UNIX или переменной окружения `TMP` в Windows. Например:

```
$tempfilename = tempnam('/tmp','data-');
print "Temporary data will be stored in $tempfilename";
Temporary data will be stored in /tmp/data-GawVoL
```

Способ, которым PHP генерирует имена временных файлов, таков, что вызов функции `tempnam()` фактически создает нужный файл, но оставляет его пустым, если сценарий не открывает этот файл непосредственно. Это гарантирует, что никакая другая программа не создаст файл с тем же именем между вызовом функции `tempnam()` и вызовом функции `fopen()` для файла с данным именем.

См. также

Документацию по функции `tmpfile()` на *http://www.php.net/tmpfile* и по функции `tempnam()` на *http://www.php.net/tempnam*.

18.3. Открытие удаленного файла

Задача

Необходимо открыть файл, доступ к которому осуществляется посредством HTTP или FTP.

Решение

Передайте URL файла функции `fopen()`:

```
$fh = fopen('http://www.example.com/robots.txt','r') or die($php_errormsg);
```

Обсуждение

Когда функции `fopen()` передается имя файла, начинающееся с *http://*, она возвращает страницу, получаемую в результате запроса HTTP/1.0 GET (чтобы обеспечить запросы к виртуальными хостам, в вызов также включается заголовок `Host`). С помощью дескриптора файла можно получить доступ только к телу ответа, но не к самому заголовку. Вызов HTTP позволяет читать файлы, но не записывать.

Если функции `fopen()` передается имя файла, начинающееся с *ftp://*, она возвращает указатель к определенному файлу, полученный посредством пассивного режима FTP. Через FTP можно открыть файл или на чтение, или на запись, но не в обоих режимах одновременно.

Чтобы открыть файл, который требует имя пользователя и пароль при использовании функции `fopen()`, вставьте идентификационную информацию в URL, как показано ниже:

```
$fh = fopen('ftp://username:password@ftp.example.com/pub/Index','r');
$fh = fopen('http://username:password@www.example.com/robots.txt','r');
```

Открытие удаленных файлов с помощью функции `fopen()` реализуется за счет внутренней функциональности PHP, называемой *URL fopen wrapper* и представляющей собою оболочку для открытия удаленных файлов. Она разрешена по умолчанию, но может быть выключена установкой параметра `allow_url_fopen` в `off` в файле *php.ini* или в файле конфигурации веб-сервера. Если не удается открыть удаленные файлы с помощью функции `fopen()`, проверьте эти настройки вашего сервера.

См. также

Рецепты с 11.1 по 11.5, в которых обсуждается извлечение URL; документацию по функции `fopen()` на *http://www.php.net/fopen* и по оболочке URL fopen wrapper на *http://www.php.net/ features.remote-files*.

18.4. Чтение из стандартного потока ввода

Задача

Необходимо читать из стандартного потока ввода.

Решение

Для открытия *php://stdin* применяется функция `fopen()`:

```
$fh = fopen('php://stdin','r') or die($php_errormsg);
while($s = fgets($fh,1024)) {
    print "You typed: $s";
}
```

Обсуждение

В рецепте 20.3 обсуждается чтение данных с клавиатуры в контексте командной строки. Чтение данных из стандартного потока ввода не очень полезно в контексте веб-интерфейса, поскольку информация из стандартного потока ввода, как таковая, не поступает в приложение. Тела HTTP-запросов POST и передаваемые для загрузки файлы предварительно анализируются средствами PHP и помещаются в специальные переменные. Их нельзя читать из стандартного потока ввода, как это делается в некоторых веб-серверах и реализациях протокола CGI.

См. также

Рецепт 20.3 о чтении с клавиатуры в контексте командной строки; документацию по функции `fopen()` на *http://www.php.net/fopen*.

18.5. Чтение файла в строку

Задача

Необходимо загрузить все содержимое файла в переменную. Например, требуется определить, соответствует ли текст в файле регулярному выражению.

Решение

Функция `filesize()` вызывается, чтобы получить размер файла, а затем надо указать функции `fread()` количество байт, которое следует прочитать:

```
$fh = fopen('people.txt','r') or die($php_errormsg);
$people = fread($fh,filesize('people.txt'));
if (preg_match('/Names:.*(David|Susannah)/i',$people)) {
    print "people.txt matches.";
}
fclose($fh) or die($php_errormsg);
```

Обсуждение

Чтобы прочитать двоичный файл (например, изображение) в Windows, необходимо добавить b к режиму открытия файла:

```
$fh = fopen('people.jpg','rb') or die($php_errormsg);
$people = fread($fh,filesize('people.jpg'));
fclose($fh);
```

Есть более короткий путь напечатать все содержимое файла, чем читать его в строку, а затем эту строку печатать. PHP предоставляет для этого две функции. Первая функция, fpassthru($fh), печатает все, что осталось в дескрипторе файла $fh, а затем закрывает его. Вторая функция, readfile($filename), сразу печатает все содержимое $filename.

Функция readfile() позволяет реализовать выборочный показ изображений, в зависимости от некоторого условия. Следующая программа требует, чтобы запрашиваемые изображения были не старше одной недели:

```
$image_directory = '/usr/local/images';

if (preg_match('/^[a-zA-Z0-9]+\.(gif|jpeg)$/',$image,$matches) &&
    is_readable($image_directory."/$image") &&
    (filemtime($image_directory."/$image") >= (time() - 86400 * 7))) {

  header('Content-Type: image/'.$matches[1]);
  header('Content-Length: '.filesize($image_directory."/$image"));

  readfile($image_directory."/$image");

} else {
  error_log("Can't serve image: $image");
}
```

Каталог, в котором хранятся изображения, $image_directory, должен находиться вне коренного каталога для документов веб-сервера, чтобы ваша программа-фильтр была эффективной. В противном случае, пользователь может просто получить непосредственный доступ к файлам изображения. Вы проверяете изображение по трем параметрам. Во-первых, проверьте, чтобы имя файла, переданное в переменной $image, состояло исключительно из букв и цифр и имело расширение *.gif* или *.jpeg*. Необходимо убедиться, что в имени файла нет таких комбинаций символов, как .. или /; это не позволит злонамеренным пользователям получить файлы, находящиеся вне указанного каталога. Во-вторых, при помощи функции is_readable() убедитесь, что у вас достаточно прав для чтения этого файла. Наконец, вызвав функцию filemtime(), получите время модификации файла и убедитесь, что это время превышает 86 400×7 секунд. В дне 86 400 секунд, поэтому значение 86 400×7 соответствует неделе.[1] Если все эти условия выполнены,

[1] При переходе от зимнего времени к летнему в дне не 86 400 секунд. Дополнительную информацию можно найти в рецепте 3.10.

то изображение готово к отправке. Сначала пошлите два заголовка, чтобы сообщить броузеру MIME-тип изображения и размер файла. Затем, вызвав функцию `readfile()`, отошлите пользователю все содержимое файла сразу.

См. также

Документацию по функции `filesize()` на *http://www.php.net/filesize*, по функции `fread()` на *http://www.php.net/ fread*, по функции `fpassthru()` на *http://www.php.net/fpassthru* и по функции `readfile()` на *http://www.php.net/ readfile*.

18.6. Подсчет строк, абзацев или записей в файле

Задача

Необходимо определить количество строк, абзацев или записей в файле.

Решение

Для подсчета строк применяется функция `fgets()`. Она читает одну строку за раз, поэтому нетрудно подсчитать, сколько раз она вызывается, пока не достигнут конец файла:

```
$lines = 0;

if ($fh = fopen('orders.txt','r')) {
  while (! feof($fh)) {
    if (fgets($fh,1048576)) {
      $lines++;
    }
  }
}
print $lines;
```

Чтобы подсчитать абзацы, увеличивайте счетчик только при чтении пустой строки:

```
$paragraphs = 0;

if ($fh = fopen('great-american-novel.txt','r')) {
  while (! feof($fh)) {
    $s = fgets($fh,1048576);
    if (("\n" == $s) || ("\r\n" == $s)) {
      $paragraphs++;
    }
  }
}
print $paragraphs;
```

Для подсчета записей увеличивайте счетчик, только когда прочитанная строка содержит разделитель записей и пробельный символ:

```
$records = 0;
$record_separator = '--end--';

if ($fh = fopen('great-american-novel.txt','r')) {
  while (! feof($fh)) {
    $s = rtrim(fgets($fh,1048576));
    if ($s == $record_separator) {
      $records++;
    }
  }
}
print $records;
```

Обсуждение

В счетчике строк переменная `$lines` увеличивает свое значение, только если функция `fgets()` возвращает истинное значение. Так как `fgets()` проходит по всему файлу, она возвращает каждую строку, которую извлекает. Достигнув последней строки, функция возвращает значение `false`, поэтому переменная `$lines` не получает неправильного приращения. Одновременно в файле будет достигнут EOF, поэтому `feof()` возвращает `true`, и цикл `while` завершается.

Этот счетчик абзацев работает прекрасно для простых текстов, но он может выдать неожиданные результаты, когда встречаются длинные или пустые строки, или когда в файле нет двух последовательных символов ограничителей строки. Эти дефекты могут быть исправлены с помощью функций, основанных на функции `preg_split()`. Если файл маленький и его можно разместить в памяти, обратитесь к функции `pc_split_paragraphs()`, показанной в примере 18.1. Она возвращает массив, содержащий каждый абзац, находящийся в файле.

Пример 18.1. pc_split_paragraphs()

```
function pc_split_paragraphs($file,$rs="\r?\n") {
    $text = join('',file($file));
    $matches = preg_split("/(.*?$rs)(?:$rs)+/s",$text,-1,
                          PREG_SPLIT_DELIM_CAPTURE|PREG_SPLIT_NO_EMPTY);
    return $matches;
}
```

Содержимое файла разбивается на две или более строк, заканчивающихся символами новой строки, и возвращается в массиве `$matches`. Разделяющее записи регулярное выражение по умолчанию, `\r?\n`, определяет разделители строк и в Windows, и в UNIX. Если же файл слишком большой, то для того чтобы прочитать его в память за один раз, вызовите функцию `pc_split_paragraphs_largefile()`, показанную в примере 18.2, которая читает файл порциями по 4 Кбайт.

Пример 18.2. pc_split_paragraphs_largefile()

```
function pc_split_paragraphs_largefile($file,$rs="\r?\n") {
    global $php_errormsg;
```

```
    $unmatched_text = '';
    $paragraphs = array();

    $fh = fopen($file,'r') or die($php_errormsg);

    while(! feof($fh)) {
        $s = fread($fh,4096) or die($php_errormsg);
        $text_to_split = $unmatched_text . $s;

        $matches = preg_split("/(.*?$rs)(?:$rs)+/s",$text_to_split,-1,
                              PREG_SPLIT_DELIM_CAPTURE|PREG_SPLIT_NO_EMPTY);

        // если последняя порция не заканчивается двумя разделителями
        // записи, то сохраните ее, чтобы разместить ее в начале
        // следующей прочитанной части
        $last_match = $matches[count($matches)-1];
        if (! preg_match("/$rs$rs\$/",$last_match)) {
            $unmatched_text = $last_match;
            array_pop($matches);
        } else {
            $unmatched_text = '';
        }

        $paragraphs = array_merge($paragraphs,$matches);
    }

    // если после чтения всех частей существует последняя порция, которая
    // не заканчивается разделителем записи, то считаем это абзацем
    if ($unmatched_text) {
        $paragraphs[] = $unmatched_text;
    }
    return $paragraphs;
}
```

Эта функция разделяет файлы на абзацы с помощью того же самого регулярного выражения, что и функция `pc_split_paragraphs()`. Когда она обнаруживает конец абзаца в порции, прочитанной из файла, она сохраняет оставшийся текст порции в переменной и размещает его в начале следующей читаемой порции. Таким образом, текст, не прошедший проверку, становится началом нового абзаца файла.

См. также

Документацию по функции `fgets()` на *http://www.php.net/fgets*, по функции `feof()` на *http://www.php.net/ feof* и по функции `preg_split()` на *http://www.php.net/preg-split*.

18.7. Обработка каждого слова в файле

Задача

Необходимо что-нибудь сделать с каждым словом, находящимся в файле.

Решение

Прочитайте каждую строку с помощью функции `fgets()`, разделите строку на слова и обработайте каждое слово:

```
$fh = fopen('great-american-novel.txt','r') or die($php_errormsg);
while (! feof($fh)) {
    if ($s = fgets($fh,1048576)) {
        $words = preg_split('/\s+/',$s,-1,PREG_SPLIT_NO_EMPTY);
        // обрабатываем слова
    }
}
fclose($fh) or die($php_errormsg);
```

Обсуждение

Ниже показано, как определить среднюю длину слова в файле:

```
$word_count = $word_length = 0;

if ($fh = fopen('great-american-novel.txt','r')) {
  while (! feof($fh)) {
    if ($s = fgets($fh,1048576)) {
      $words = preg_split('/\s+/',$s,-1,PREG_SPLIT_NO_EMPTY);
      foreach ($words as $word) {
        $word_count++;
        $word_length += strlen($word);
      }
    }
  }
}

print sprintf("The average word length over %d words is %.02f characters.",
              $word_count,
              $word_length/$word_count);
```

Обработка каждого слова происходит по-разному, в зависимости от того, как определяется понятие «слово». Программа в этом рецепте для Perl-совместимых регулярных выражений использует пробельный метасимвол `\s`, который может быть пробелом, символом табуляции, символом новой строки, возвратом каретки и переводом страницы. Рецепт 1.5 разбивает строку на слова, выделяя пробел, что полезно в этом рецепте, т. к. слова должны быть снова соединены с пробелами. Perl-диалект регулярных выражений содержит специальный квантификатор для границы слова, который сравнивает между собой словарные символы (буквенно-цифровые и символ подчеркивания) и не словарные символы (какие-нибудь другие). Разграничение слов посредством `\b` вместо `\s` существенно изменяет трактовку слов, содержащих знаки пунктуации. Выражение `6 o'clock` будет состоять из двух слов, если для их разделения будут использоваться пробельные символы (`6` и `o'clock`), и оно же распадается на четыре слова, если они разделяются символом границы слова (`6`, `o`, `'` и `clock`).

См. также

Рецепт 13.2, в котором обсуждаются регулярные выражения для сравнения слов; рецепт 1.4 о разбиении строки на слова; документацию по функции `fgets()` на *http://www.php.net/fgets*, по функции `preg_split()` на *http://www.php.net/preg-split* и материалы о Perl-совместимом расширении регулярных выражений на *http://www.php.net/pcre*.

18.8. Чтение определенной строки в файле

Задача

Необходимо прочитать из файла определенную строку, например самую свежую запись в гостевой книге, которая была добавлена в конец файла гостевой книги.

Решение

Если файл помещается в память, то прочитайте его в массив и извлеките соответствующий элемент массива:

```
$lines = file('vacation-hotspots.txt');
print $lines[2];
```

Обсуждение

Поскольку индексирование массива начинается с 0, то элемент `$lines[2]` ссылается на третью строку файла.

Если файл не помещается в память, то читайте его строку за строкой и отслеживайте, на какой строке вы находитесь:

```
$line_counter = 0;
$desired_line = 29;

$fh = fopen('vacation-hotspots.txt','r') or die($php_errormsg);
while ((! feof($fh)) && ($line_counter <= $desired_line)) {
    if ($s = fgets($fh,1048576)) {
        $line_counter++;
    }
}
fclose($fh) or die($php_errormsg);

print $s;
```

Если продолжить логику кода из раздела «Решение», то в результате присваивания `$desired_line = 29` будет выведена 30-я строка файла. Для печати 29-й строки файла измените цикл `while` следующим образом:

```
while ((! feof($fh)) && ($line_counter < $desired_line)) {
```

См. также

Документацию по функции `fgets()` на *http://www.php.net/fgets* и по функции `feof()` на *http://www.php.net/ feof*.

18.9. Обработка файла по строкам или абзацам в обратном направлении

Задача

Необходимо осуществить некие действия над каждой строкой файла, начиная с конца. Например, нетрудно добавить новую запись гостевой книги в конец файла, открыв его в режиме добавления, но при показе требуется сначала отобразить самые последние записи, поэтому необходимо обрабатывать файл, начиная с конца.

Решение

Если файл помещается в памяти, то надо с помощью функции `file()` прочитать каждую строку файла в массив, а затем перевернуть этот массив:

```
$lines = file('guestbook.txt');
$lines = array_reverse($lines);
```

Обсуждение

Можно также выполнить цикл и по не перевернутому массиву строк, начиная с конца. Ниже показано, как напечатать последние 10 строк файла, начиная с последней строки:

```
$lines = file('guestbook.txt');
for ($i = 0, $j = count($lines); $i <= 10; $i++) {
    print $lines[$j - $i];
}
```

См. также

Документацию по функции `file()` на *http://www.php.net/file* и по функции `array_reverse()` на *http://www.php.net/array-reverse*.

18.10. Выбор случайной строки из файла

Задача

Необходимо выбрать из файла случайную строку, например показать выборку из файла афоризмов.

Решение

Вызовите функцию `pc_randomint()`, показанную в примере 18.3, которая распределяет возможность выбора равномерно по всем строкам файла.

Пример 18.3. pc_randomint( )

```
function pc_randomint($max = 1) {
  $m = 1000000;
  return ((mt_rand(1,$m * $max)-1)/$m);
}
```

Приведем пример применения функции `pc_randomint()`:

```
$line_number = 0;

$fh = fopen('sayings.txt','r') or die($php_errormsg);
while (! feof($fh)) {
    if ($s = fgets($fh,1048576)) {
        $line_number++;
        if (pc_randomint($line_number) < 1) {
            $line = $s;
        }
    }
}
fclose($fh) or die($php_errormsg);
```

Обсуждение

Функция `pc_randomint()` вычисляет случайное десятичное число между 0 и `$max`, включая 0, но исключая `$max`. После чтения каждой строки счетчик строк увеличивается, а функция `pc_randomint()` генерирует случайное число между 0 и `$line_number`. Если число меньше 1, то в качестве случайной строки выбирается текущая строка. После того как прочитаны все строки, последняя выбранная случайная строка остается в переменной `$line`.

Этот алгоритм практически обеспечивает вероятность выбора каждой строки файла из n строк, равную $1/n$, без необходимости запоминать все строки в памяти.

См. также

Документацию по функции `mt_rand()` на *http://www.php.net/mt-rand*.

18.11. Рандомизация всех строк в файле

Задача

Необходимо выполнить случайную перестановку всех строк в файле. Например, есть файл забавных цитат, и вы хотите выбрать из них случайную.

Решение

Прочитайте все строки файла в массив с помощью функции `file()`, а затем перетасуйте элементы массива:

```
$lines = file('quotes-of-the-day.txt');
$lines = pc_array_shuffle($lines);
```

Обсуждение

Функция `pc_array_shuffle()` из рецепта 4.20 дает более случайный результат, чем встроенная в PHP функция `shuffle()`, поскольку она основана на алгоритме перемешивания Фишера-Йетса (Fisher-Yates), равномерно распределяющем элементы внутри массива.

См. также

Рецепт 4.19 о функции `pc_array_shuffle()`; документацию по функции `shuffle()` на *http://www.php.net/ shuffle*.

18.12. Обработка текстовых полей переменной длины

Задача

Необходимо прочитать из файла разграниченные текстовые поля. Например, есть программа базы данных, построчно выводящая записи, в которых поля разделены символами табуляции, и требуется преобразовать эти данные в массив.

Решение

Читайте каждую строку, а затем выделяйте поля на основе их символа разделителя:

```
$delim = '|';

$fh = fopen('books.txt','r') or die("can't open: $php_errormsg");
while (! feof($fh)) {
    $s = rtrim(fgets($fh,1024));
    $fields = explode($delim,$s);
    // ... делаем что-нибудь с данными ...
}
fclose($fh) or die("can't close: $php_errormsg");
```

Обсуждение

Разберем следующие данные из файла *books.txt*:

```
Elmer Gantry|Sinclair Lewis|1927
The Scarlatti Inheritance|Robert Ludlum|1971
```

```
The Parsifal Mosaic|Robert Ludlum|1982
Sophie's Choice|William Styron|1979
```

Обработаем каждую запись следующим образом:

```
$fh = fopen('books.txt','r') or die("can't open: $php_errormsg");
while (! feof($fh)) {
    $s = rtrim(fgets($fh,1024));
    list($title,$author,$publication_year) = explode('|',$s);
    // ... делаем что-нибудь с данными ...
}
fclose($fh) or die("can't close: $php_errormsg");
```

Аргумент длины строки в функции `fgets()` должен быть как минимум равен длине самой большой записи, чтобы эта запись не была усечена.

Вызов функции `rtrim()` необходим, поскольку функция `fgets()` вставляет завершающий пробельный символ в строку, которую она читает. Без функции `rtrim()` в конце каждой переменной `$publication_year` находился бы символ новой строки.

См. также

Рецепт 1.11, в котором обсуждаются способы разбиения строк на части; рецепты 1.9 и 1.10, посвященные анализу данных, разделенных запятыми, и данных с фиксированной шириной полей; документацию по функции `explode()` на *http://www.php.net/explode* и по функции `rtrim()` на *http://www.php.net/rtrim*.

18.13. Чтение файлов конфигурации

Задача

Инициализации настроек в программах выполняется с помощью конфигурационных файлов.

Решение

Это делается при помощи функции `parse_ini_file()`:

```
$config = parse_ini_file('/etc/myapp.ini');
```

Обсуждение

Функция `parse_ini_file()` читает файлы конфигурации, структурированные примерно так же, как основной файл PHP, *php.ini*. Однако функция `parse_ini_file()` возвращает значения из файла в массив, при этом настройки конфигурационного файла не влияют на конфигурирование PHP.

Например, если функция `parse_ini_file()` получает файл с таким содержанием:

```
; physical features
```

```
eyes=brown
hair=brown
glasses=yes

; other features
name=Susannah
likes=monkeys,ice cream,reading
```

Она возвращает следующий массив:

```
Array
(
    [eyes] => brown
    [hair] => brown
    [glasses] => 1
    [name] => Susannah
    [likes] => monkeys,ice cream,reading
)
```

Пустые строки и строки файла конфигурации, начинающиеся с `;`, игнорируются. Остальные строки с парами `name=value` помещаются в массив с именем в качестве ключа и, соответственно, со значением в качестве значения. Такие слова, как `on` и `yes`, в качестве значений возвращаются в виде 1, а такие слова, как `off` и `no`, возвращаются в виде пустой строки.

Чтобы проанализировать разделы файла конфигурации, передайте `1` функции `parse_ini_file()` в качестве второго аргумента. Разделы – это множество заключенных в квадратные скобки слов в файле:

```
[physical]
eyes=brown
hair=brown
glasses=yes

[other]
name=Susannah
likes=monkeys,ice cream,reading
```

Если это находится в файле */etc/myapp.ini*, то:

```
$conf = parse_ini_file('/etc/myapp.ini',1);
```

Помещает следующий массив в переменную `$conf`:

```
Array
(
    [physical] => Array
        (
            [eyes] => brown
            [hair] => brown
            [glasses] => 1
        )

    [other] => Array
        (
            [name] => Susannah
```

```
                [likes] => monkeys,ice cream,reading
            )

    )
```

Конфигурационный файл может быть также самостоятельным PHP-файлом, загружаемым с помощью `require` вместо функции `parse_ini_file()`. Если файл *config.php* содержит:

```
<?php

// физические данные
$eyes = 'brown';
$hair = 'brown';
$glasses = 'yes';

// другие данные
$name = 'Susannah';
$likes = array('monkeys','ice cream','reading');
?>
```

Переменные `$eyes`, `$hair`, `$glasses`, `$name` и `$likes` можно установить так:

```
require 'config.php';
```

Конфигурационный файл, загруженный с помощью `require`, должен быть полноценным файлом PHP, включая начальный тег `<?php` и заключительный тег `?>`. Переменные, указанные в файле *config.php*, устанавливаются явным образом, а не внутри массива, как в функции `parse_ini_file()`. Для простых файлов конфигурации эта методика может не стоить такого дополнительного внимания к синтаксису, но она полезна для внедрения логики в конфигурационный файл:

```
<?php

$time_of_day = (date('a') == 'am') ? 'early' : 'late';

?>
```

Возможность внедрять логику в конфигурационные файлы стоит того, чтобы создавать файлы с кодом PHP, но полезно также иметь в массиве и все множество переменных файла конфигурации. Новые версии PHP будут снабжены функциональностью, называемой *пространства имен*, которая позволит иерархически объединять переменные в различные группы; переменная `$hair` в двух разных пространствах имен может иметь два разных значения. С помощью пространств имен все значения конфигурационного файла могут быть загружены в пространство имен `Config`, так чтобы они не пересекались с другими переменными.

См. также

Документацию по функции `parse_ini_file()` на *http://www.php.net/parse-ini-file*; информацию о пространствах имен и других новых возможностях языка PHP, доступную на *http://www.php.net/ZEND_CHANGES.txt*.

18.14. Чтение или запись в определенное место в файле

Задача

Необходимо читать из определенного места в файле (или записывать в него). Например, надо заменить третью запись в файле, состоящем из записей длиной 80 байт, поэтому требуется начинать запись со 161-го байта.

Решение

Функция `fseek()` позволяет переместиться на определенное количество байт, считая с начала файла, с конца файла или с текущей позиции в файле:

```
fseek($fh,26);            // 26 байт, считая с начала файла
fseek($fh,26,SEEK_SET);   // 26 байт, считая с начала файла
fseek($fh,-39,SEEK_END);  // 39 перед концом файла
fseek($fh,10,SEEK_CUR);   // 10 вперед, начиная с текущей позиции
fseek($fh,0);             // начало файла
```

Функция `rewind()` перемещается в начало файла:

```
rewind($fh);              // то же самое, что и fseek($fh,0)
```

Обсуждение

Функция `fseek()` возвращает 0, если она может перейти в указанную позицию, в противном случае она возвращает –1. Поиск за концом файла не является ошибкой для функции `fseek()`. А функция `rewind()`, столкнувшись с ошибкой, возвращает 0.

Функция `fseek()` может работать только с локальными файлами и не может с файлами, открытыми функцией `fopen()` по http или FTP. Если функции `fseek()` передать дескриптор удаленного файла, то она выдаст ошибку `E_NOTICE`.

Для получения текущей позиции в файле применяется функция `ftell()`:

```
if (0 === ftell($fh)) {
  print "At the beginning of the file.";
}
```

Функция `ftell()` в случае ошибки возвращает `false`, поэтому необходим оператор `===` – для того, чтобы гарантировать, что ее возвращаемое значение действительно представляет целочисленный 0.

См. также

Документацию по функции `fseek()` на *http://www.php.net/fseek*, по функции `ftell()` на *http://www.php.net/ftell* и по функции `rewind()` на *http://www.php.net/rewind*.

18.15. Удаление из файла последней строки

Задача

Необходимо удалить последнюю строку файла. Например, кто-то добавил комментарий в конце вашей гостевой книги, который вам не нравится, поэтому вы хотите от него избавиться.

Решение

Если файл маленький, то можно прочитать его в массив с помощью функции `file()`, а затем удалить последний элемент массива:

```
$lines = file('employees.txt');
array_pop($lines);
$file = join('',$lines);
```

Обсуждение

Если файл большой, то для чтения его в массив требуется слишком много памяти. В этом случае нужна программа, отыскивающая конец файла и работающая в обратном направлении, останавливаясь при обнаружении символа новой строки:

```
$fh = fopen('employees.txt','r') or die("can't open: $php_errormsg");
$linebreak = $beginning_of_file = 0;

$gap = 80;
$filesize = filesize('employees.txt');
fseek($fh,0,SEEK_END);

while (! ($linebreak || $beginning_of_file)) {
    // записываем в то место файла, где находимся
    $pos = ftell($fh);

    /* возвращаемся на $gap символов, вызываем функцию rewind(),
     * чтобы перейти в начало, если мы находимся в позиции
     * меньше $gap символов от начала файла */
    if ($pos < $gap) {
        rewind($fh);
    } else {
        fseek($fh,-$gap,SEEK_CUR);
    }

    // читаем $gap символов, которые мы только что нашли, двигаясь назад
    $s = fread($fh,$gap) or die($php_errormsg);

    /* если мы читаем по направлению к концу файла,
     * то удаляем последний символ, так как если это символ
     * новой строки, то мы должны игнорировать его */
    if ($pos + $gap >= $filesize) {
        $s = substr_replace($s,'',-1);
    }
```

```
        // переходим обратно на то место, где мы были до чтения $gap символов в $s
        if ($pos < $gap) {
            rewind($fh);
        } else {
            fseek($fh,-$gap,SEEK_CUR);
        }

        // присутствует ли в $s символ конца строки?
        if (is_integer($lb = strrpos($s,"\n"))) {
            $linebreak = 1;
            // новая строка файла начинается сразу за символом конца строки
            $line_end = ftell($fh) + $lb + 1;
        }

        // выходим из цикла, если мы находимся в начале файла
        if (ftell($fh) == 0) { $beginning_of_file = 1; }

    }
    if ($linebreak) {
        rewind($fh);
        $file_without_last_line = fread($fh,$line_end) or die($php_errormsg);
    }
    fclose($fh) or die("can't close: $php_errormsg");
```

Эта программа начинает с конца файла и перемещается в обратном направлении порциями по `$gap` символов в поисках символа новой строки. Найдя его, она узнает последнюю строку файла, которая начинается сразу после символа новой строки. Эта позиция сохраняется в переменной `$line_end`. После цикла `while`, если переменная `$linebreak` установлена, содержимое файла от начала до `$line_end` читается в переменную `$file_without_last_line`.

Последний символ файла игнорируется, поскольку если это символ новой строки, то он не означает начало последней строки файла. Рассмотрим 10-символьный файл, содержимое которого состоит из символов `asparagus\n`. В нем только одна строка, состоящая из слова `asparagus` и символа новой строки. Без своей последней строки этот файл пуст, о чем и сообщит предыдущая программа. Если бы она начала сканирование с последнего символа, то увидела бы символ новой строки и вышла бы из цикла сканирования, неправильно выведя строку `asparagus` без символа новой строки.

См. также

Рецепт 18.14, в котором более подробно обсуждаются функции `fseek()` и `rewind()`; документацию по функции `array_pop()` на *http://www.php.net/array-pop*, по функции `fseek()` на *http://www.php.net/fseek* и по функции `rewind()` на *http://www.php.net/rewind*.

18.16. Непосредственная модификация файла без временной копии

Задача

Необходимо изменить файл, не создавая временный файл для сохранения изменений.

Решение

Прочитайте файл в память, выполните изменения и перезапишите файл. Откройте файл в режиме `r+` (в Windows `rb+`, если необходимо) и откорректируйте его длину с помощью функции `ftruncate()` после записи изменений:

```
// открываем файл на чтение и запись
$fh = fopen('pickles.txt','r+')         or die($php_errormsg);

// читаем весь файл в переменную $s
$s = fread($fh,filesize('pickles.txt')) or die($php_errormsg);

// ... модифицируем $s ...

// ищем начало файла и записываем новую переменную $s
rewind($fh);
if (-1 == fwrite($fh,$s))                { die($php_errormsg); }

// подгоняем длину файла к тому, что было записано
ftruncate($fh,ftell($fh))               or die($php_errormsg);

// закрываем файл
fclose($fh)                             or die($php_errormsg);
```

Обсуждение

Следующая программа превращает текст, выделенный звездочками или косыми чертами, в текст с тегами HTML `<b>` или `<i>`:

```
$fh = fopen('message.txt','r+')         or die($php_errormsg);

// читаем весь файл в переменную $s
$s = fread($fh,filesize('message.txt')) or die($php_errormsg);

// преобразуем *word* to <b>word</b>
$s = preg_replace('@\*(.*?)\*@i','<b>$1</b>',$s);
// преобразуем /word/ to <i>word</i>
$s = preg_replace('@/(.*?)/@i','<i>$1</i>',$s);

rewind($fh);
if (-1 == fwrite($fh,$s))                { die($php_errormsg); }
ftruncate($fh,ftell($fh))               or die($php_errormsg);
fclose($fh)                             or die($php_errormsg);
```

Поскольку добавление тегов HTML приводит к росту файла, то весь файл должен быть прочитан в память, а затем обработан. Если измене-

ния в файле уменьшают длину каждой строки (или оставляют тот же размер), то файл можно обработать строку за строкой, сэкономив память. Следующий пример преобразует текст, размеченный тегами <b> и <i>, в текст, выделенный звездочками и косыми чертами:

```
$fh = fopen('message.txt','r+')         or die($php_errormsg);

// определяем количество байт для чтения
$bytes_to_read = filesize('message.txt');

// инициализируем переменные, хранящие позиции в файле
$next_read = $last_write = 0;

// продолжаем, пока есть байты для чтения
while ($next_read < $bytes_to_read) {

    /* переходим в следующую позицию чтения, читаем строку и сохраняем
     * позицию для следующего чтения */
    fseek($fh,$next_read);
    $s = fgets($fh,1048576)             or die($php_errormsg);
    $next_read = ftell($fh);

    // преобразуем <b>word</b> в *word*
    $s = preg_replace('@<b[^>]*>(.*?)</b>@i','*$1*',$s);
    // преобразуем <i>word</i> в /word/
    $s = preg_replace('@<i[^>]*>(.*?)</i>@i','/$1/',$s);

    /* переходим в позицию, где закончилась предыдущая запись, записываем
     * преобразованную строку и сохраняем позицию для следующей записи */
    fseek($fh,$last_write);
    if (-1 == fwrite($fh,$s))            { die($php_errormsg); }
    $last_write = ftell($fh);
}

// сокращаем длину файла до величины прочитанного
ftruncate($fh,$last_write)              or die($php_errormsg);

// закрываем файл
fclose($fh)                             or die($php_errormsg);
```

См. также

Дополнительную информацию о преобразовании между ASCII и HTML в рецептах 11.9 и 11.10; рецепт 18.14, в котором более подробно обсуждаются функции `fseek()` и `rewind()`; документацию по функции `fseek()` на *http://www.php.net/fseek*, по функции `rewind()` на *http://www.php.net/rewind* и по функции `ftruncate()` на *http://www.php.net/ftruncate*.

18.17. Сброс вывода в файл

Задача

Необходимо принудительно сбросить всю информацию из буфера в файл с нужным дескриптором.

Решение

Вызовите функцию `fflush()`:

```
fwrite($fh,'There are twelve pumpkins in my house.');
fflush($fh);
```

Это гарантирует, что строка «There are twelve pumpkins in my house.» записывается в файл с дескриптором `$fh`.

Обсуждение

Из соображений эффективности системные библиотеки ввода/вывода в большинстве случаев не записывают в файл, когда получают такую команду. Вместо этого они накапливают записи в буфере и записывают их все на диск одновременно. Применение функции `fflush()` форсирует фактическую запись информации, ожидающей в буфере, на диск.

Сброс вывода может быть особенно полезен при создании журнала доступа или журнала активности. Вызов функции `fflush()` после каждого сообщения в файл журнала гарантирует, что любой (человек или программа), кто следит за файлом журнала, увидит сообщение так скоро, как это возможно.

См. также

Документацию по функции `fflush()` на *http://www.php.net/fflush*.

18.18. Запись в стандартный поток вывода

Задача

Необходимо записать в стандартный поток вывода.

Решение

Примените `echo` или `print`:

```
print "Where did my pastrami sandwich go?";
echo  "It went into my stomach.";
```

Обсуждение

В то время как `print()` — это функция, `echo` представляет собой конструкцию языка. Это означает, что функция `print()` возвращает значение, тогда как `echo` нет. Это значит, что можно включить в выражение функцию `print()`, но не `echo`:

```
// это правильно
(12 == $status) ? print 'Status is good' : error_log('Problem with status!');

// а это приведет к синтаксической ошибке
(12 == $status) ? echo 'Status is good' : error_log('Problem with status!');
```

Используйте *php://stdout* в качестве имени файла, если вы применяете файловые функции:

```
$fh = fopen('php://stdout','w') or die($php_errormsg);
```

Записывать в стандартный поток вывода посредством дескриптора файла, вместо того чтобы просто применить `echo` или функцию `print()`, полезно, если необходимо абстрагироваться от того, куда идет ваш вывод, или осуществить стандартный вывод одновременно с записью в файл. Подробности можно найти в рецепте 18.19.

Можно также писать в стандартный поток ошибок, открыв *php://stderr*:

```
$fh = fopen('php://stderr','w');
```

См. также

Рецепт 18.19 об одновременной записи в несколько файловых дескрипторов; документацию по функции `echo` на *http://www.php.net/echo* и по функции `print()` на *http://www.php.net/print*.

18.19. Запись в несколько файловых дескрипторов одновременно

Задача

Необходимо направить вывод в более чем один дескриптор файла; например, требуется регистрировать сообщения на экране и в файле.

Решение

Заключите ваш вывод в цикл по файловым дескрипторам, как показано в примере 18.4.

Пример 18.4. pc_multi_fwrite( )

```
function pc_multi_fwrite($fhs,$s,$length=NULL) {
  if (is_array($fhs)) {
    if (is_null($length)) {
      foreach($fhs as $fh) {
        fwrite($fh,$s);
      }
    } else {
      foreach($fhs as $fh) {
        fwrite($fh,$s,$length);
      }
    }
  }
}
```

Приведем пример:

```
$fhs['file'] = fopen('log.txt','w') or die($php_errormsg);
```

```
$fhs['screen'] = fopen('php://stdout','w') or die($php_errormsg);

pc_multi_fwrite($fhs,'The space shuttle has landed.');
```

Обсуждение

Если вы не хотите передавать аргумент длины в функцию `fwrite()` (или вы не всегда хотите это делать), можно исключить эту проверку из функции `pc_multi_fwrite()`. Следующая версия не принимает аргумент `$length`:

```
function pc_multi_fwrite($fhs,$s) {
  if (is_array($fhs)) {
    foreach($fhs as $fh) {
      fwrite($fh,$s);
    }
  }
}
```

См. также

Документацию по функции `fwrite()` на *http://www.php.net/fwrite*.

18.20. Преобразование метасимволов среды в escape-последовательности

Задача

Необходимо включить в командную строку внешние данные, например передать в программу пользовательский ввод в качестве аргумента, но для предотвращения всяких неожиданностей требуется преобразовать специальные символы в escape-последовательности.

Решение

Обработка аргументов выполняется посредством функции `escapeshellarg()`:

```
system('ls -al '.escapeshellarg($directory));
```

А функция `escapeshellcmd()` применяется для обработки имен программ:

```
system(escapeshellcmd($ls_program).' -al');
```

Обсуждение

Рискованно помещать в командную строку символы, не преобразованные в escape-последовательности. Никогда не передавайте непреобразованный пользовательский ввод ни в одну из функций PHP, работающих с командной средой. Всегда преобразуйте в escape-последовательности соответствующие символы в команде и в аргументах. Это очень

важно. Не принято выполнять командные строки, поступившие из веб-форм, и мы даже в шутку не рекомендуем ничего подобного. Однако иногда требуется запустить внешнюю программу, поэтому преобразование команд и аргументов в escape-последовательности полезно.

Функция escapeshellarg() заключает аргументы в одиночные кавычки (и превращает в escape-последовательность каждую стоящую особняком одиночную кавычку). Следующий код позволяет вывести статус определенного процесса:

```
system('/bin/ps '.escapeshellarg($process_id));
```

Вызов функции escapeshellarg() гарантирует, что будет отображен соответствующий процесс, даже если он содержит неожиданные символы (например, пробел). Это также предотвращает запуск непредусмотренных команд. Если переменная $process_id содержит:

```
1; rm -rf /
```

то:

```
system("/bin/ps $process_id")
```

не только показывает статус процесса, равный 1, но также выполняет команду *rm -rf /*. Однако:

```
system('/bin/ps '.escapeshellarg($process_id))
```

выполняет команду */bin/ps 1; rm -rf*, которая выдает ошибку, поскольку строка «1-точка с запятой-пробел-rm-пробел-дефис-rf» не является допустимым идентификатором процесса.

Точно так же, escapeshellcmd() предотвращает выполнение непредусмотренных командных строк. Этот код запускает различные программы, в зависимости от значения $which_program:

```
system("/usr/local/bin/formatter-$which_program");
```

Например, если значение переменной $which_program равно pdf 12, то сценарий запустит команду */usr/local/bin/formatter-pdf* с аргументом 12. Но если переменная $which_program равна pdf 12; 56, то сценарий запустит */usr/local/ bin/formatter-pdf* с аргументом 12, а затем запустит программу *56*, что является ошибкой. Чтобы успешно передать аргументы программе *formatter-pdf*, необходима функция escapeshellcmd():

```
system(escapeshellcmd("/usr/local/bin/formatter-$which_program"));
```

Этот код запускает программу */usr/local/bin/formatter-pdf* и передает ей два аргумента — 12 и 56.

См. также

Документацию по функции system() на *http://www.php.net/system*, по функции escapeshellarg() на *http://www.php.net/escapeshellarg* и по функции escapeshellcmd() на *http://www.php.net/escapeshellcmd*.

18.21. Передача входной информации в программу

Задача

Пусть надо передать внешней программе входную информацию из сценария на PHP. Например, при работе с базой данных вам потребовалось запустить внешнюю программу для индексирования текста, и этой программе надо передать текст.

Решение

Откройте канал к программе с помощью функции `popen()`, выполните запись в канал с помощью функций `fputs()` или `fwrite()`, затем закройте канал, вызвав функцию `pclose()`:

```
$ph = popen('program arg1 arg2','w')          or die($php_errormsg);
if (-1 == fputs($ph,"first line of input\n"))  { die($php_errormsg); }
if (-1 == fputs($ph,"second line of input\n")) { die($php_errormsg); }
pclose($ph)                                   or die($php_errormsg);
```

Обсуждение

Этот пример использует функцию `popen()` для вызова команды *nsupdate*, которая посылает серверу имен запросы динамического обновления DNS (Dynamic DNS Update):

```
$ph = popen('/usr/bin/nsupdate -k keyfile')   or die($php_errormsg);
if (-1 == fputs($ph,"update delete test.example.com A\n"))
                { die($php_errormsg); }
if (-1 == fputs($ph,"update add test.example.com 5 A 192.168.1.1\n"))
                { die($php_errormsg); }
pclose($ph)                                   or die($php_errormsg);
```

Две команды посылаются *nsupdate* посредством функции `popen()`. Первая удаляет A-запись *test.example.com*, а вторая добавляет новую A-запись для *test.example.com* с адресом 192.168.1.1.

См. также

Документацию по функции `popen()` на *http://www.php.net/popen* и по функции `pclose()` на *http://www.php.net/pclose*; Динамический DNS, описанный в RFC 2136 на *http://www.faqs.org/ rfcs/rfc2136.html*.

18.22. Чтение из стандартного потока вывода программы

Задача

Необходимо прочитать стандартный поток вывода программы; например, требуется вывод системной утилиты, такой как *route(8)*, предоставляющей сетевую информацию.

Решение

Для того чтобы прочитать все содержимое программного вывода, применяется оператор обратного апострофа (`):

```
$routing_table = `/sbin/route`;
```

Чтобы читать вывод частями, откройте канал к функции `popen()`:

```
$ph = popen(`/sbin/route`,`r`) or die($php_errormsg);
while (! feof($ph)) {
    $s = fgets($ph,1048576)    or die($php_errormsg);
}
pclose($ph)                    or die($php_errormsg);
```

Обсуждение

Оператор обратного апострофа (который недоступен в безопасном режиме) выполняет программу и возвращает ее вывод одной строкой. На машине под управлением Linux с оперативной памятью объемом 448 Мбайт следующая команда:

```
$s = `/usr/bin/free`;
```

помещает в переменную `$s` следующий многострочный вывод:

```
             total       used       free     shared    buffers     cached
Mem:        448620     446384       2236          0      68568     163040
-/+ buffers/cache:     214776     233844
Swap:       136512          0     136512
```

Если программа генерирует большой вывод, то с точки зрения эффективности распределения памяти следует читать из канала по одной строке за раз. Если вы передаете форматированную информацию броузеру, основываясь на выводе канала, то можно передавать данные по мере их поступления. Следующий пример выводит информацию о недавних регистрациях в системе UNIX, отформатированную в виде HTML-таблицы. В нем используется команда */usr/bin/last*:

```
// печатаем заголовок таблицы
print<<<_HTML_
<table>
<tr>
 <td>user</td><td>login port</td><td>login from</td><td>login time</td>
 <td>time spent logged in</td>
</tr>
_HTML_;

// открываем канал к /usr/bin/last
$ph = popen(`/usr/bin/last`,`r`) or die($php_errormsg);
while (! feof($ph)) {
    $line = fgets($ph,80) or die($php_errormsg);

    // не обрабатываем пустые строки или информационную строку в конце
```

```
        if (trim($line) && (! preg_match('/^wtmp begins/',$line))) {
            $user = trim(substr($line,0,8));
            $port = trim(substr($line,9,12));
            $host = trim(substr($line,22,16));
            $date = trim(substr($line,38,25));
            $elapsed = trim(substr($line,63,10),' ()');

            if ('logged in' == $elapsed) {
                $elapsed = 'still logged in';
                $date = substr_replace($date,'',-5);
            }

            print "<tr><td>$user</td><td>$port</td><td>$host</td>";
            print "<td>$date</td><td>$elapsed</td></tr>\n";
        }
    }
    pclose($ph) or die($php_errormsg);

    print '</table>';
```

См. также

Документацию по функции `popen()` на *http://www.php.net/popen*, по функции `pclose()` на *http://www.php.net/pclose* и по оператору обратного апострофа на *http://www.php.net/language.operators.execution*; о безопасном режиме на *http://www.php.net/features.safe-mode*.

18.23. Чтение из стандартного потока ошибок программы

Задача

Необходимо читать программный вывод ошибок. Например, требуется перехватить системные вызовы, показываемые *strace(1)*.

Решение

Перенаправьте стандартный поток ошибок в стандартный поток вывода, добавив `2>&1` в командную строку, передаваемую функции `popen()`. Прочитайте стандартный поток вывода, открыв канал в режиме `r`:

```
$ph = popen('strace ls 2>&1','r') or die($php_errormsg);
while (!feof($ph)) {
    $s = fgets($ph,1048576)       or die($php_errormsg);
}
pclose($ph)                       or die($php_errormsg);
```

Обсуждение

И в оболочке UNIX *sh*, и в оболочке Windows *cmd.exe* стандартный поток ошибок представляет файловый дескриптор 2, а стандартный поток вывода – файловый дескриптор 1. Добавление `2>&1` в командную

строку приказывает оболочке перенаправить то, что обычно направляется в файловый дескриптор 2 (стандартный поток ошибок), в файловый дескриптор 1 (стандартный поток вывода). Затем функция `fgets()` читает и стандартный поток ошибок, и стандартный поток вывода.

Этот способ позволяет читать стандартный поток ошибок, но не дает возможности отличить его от стандартного потока вывода. Для того чтобы читать исключительно стандартный поток ошибок, нельзя допустить возвращение стандартного потока вывода через канал. Это делается путем перенаправления его в */dev/null* в UNIX и в NUL в Windows:

```
// UNIX: читаем только стандартный поток ошибок
$ph = popen('strace ls 2>&1 1>/dev/null','r') or die($php_errormsg);
// Windows: читаем только стандартный поток ошибок
$ph = popen('ipxroute.exe 2>&1 1>NUL','r') or die($php_errormsg);
```

См. также

Документацию по функции `popen()` на *http://www.php.net/popen*; подробную информацию об оболочке, используемой вашей системой в функции `popen()`, см. на справочной странице man *popen(3)*; информацию о перенаправлении оболочки в системе UNIX можно найти в разделе *Redirection* справочной страницы *sh(1)* программы *man*; в Windows – описание перенаправления в разделе справочной системы, посвященном командам (Command Reference).

18.24. Блокировка файла

Задача

Необходимо получить исключительный доступ к файлу, чтобы не допустить его изменения, пока вы читаете или обновляете этот файл. Например, данные гостевой книги записываются в файл, тогда два пользователя должны иметь возможность одновременно добавлять записи в гостевую книгу, не уничтожая информацию друг друга.

Решение

Вызовите функцию `flock()` для выполнения консультативной блокировки:

```
$fh = fopen('guestbook.txt','a')         or die($php_errormsg);
flock($fh,LOCK_EX)                       or die($php_errormsg);
fwrite($fh,$_REQUEST['guestbook_entry']) or die($php_errormsg);
fflush($fh)                              or die($php_errormsg);
flock($fh,LOCK_UN)                       or die($php_errormsg);
fclose($fh)                              or die($php_errormsg);
```

Обсуждение

Блокировка файла, устанавливаемая функцией `flock()`, называется *консультативной* (*advisory*), поскольку в действительности функция `flock()` не запрещает другим процессам открывать заблокированный файл, она лишь предоставляет способ добровольного сотрудничества при доступе к файлу. Все программы, которым нужен доступ к файлам, блокирующимся с помощью функции `flock()`, должны также устанавливать и снимать блокировку, чтобы сделать блокировку файла эффективной.

С помощью функции `flock()` можно устанавливать блокировку двух типов: монопольную и разделяемую. *Монопольная блокировка* (*exclusive lock*), определяемая константой `LOCK_EX` в качестве второго аргумента функции `flock()`, может быть удержана одновременно для конкретного файла только одним процессом. *Разделяемая блокировка* (*shared lock*), определяемая константой `LOCK_SH`, может быть удержана одновременно для конкретного файла более чем одним процессом. Перед записью в файл необходимо установить монопольную блокировку. Перед чтением из файла необходимо установить разделяемую блокировку.

Чтобы разблокировать файл, вызовите функцию `flock()` со значением `LOCK_UN` в качестве второго аргумента. Важно сбросить все данные буфера, которые должны быть записаны в файл, до того, как файл будет разблокирован. Другие процессы не получат возможность блокировки, пока данные не будут записаны.

По умолчанию функция `flock()` блокирует, если имеет такую возможность. Чтобы приказать ей не блокировать, добавьте ко второму аргументу значение `LOCK_NB`:

```
$fh = fopen('guestbook.txt','a')         or die($php_errormsg);
$tries = 3;
while ($tries > 0) {
    $locked = flock($fh,LOCK_EX | LOCK_NB);
    if (! $locked) {
        sleep(5);
        $tries--;
    } else {
        // не выполнять цикл повторно
        $tries = 0;
    }
}
if ($locked) {
    fwrite($fh,$_REQUEST['guestbook_entry']) or die($php_errormsg);
    fflush($fh)                              or die($php_errormsg);
    flock($fh,LOCK_UN)                       or die($php_errormsg);
    fclose($fh)                              or die($php_errormsg);
} else {
    print "Can't get lock.";
}
```

В режиме отмены блокировки функция `flock()` выходит немедленно, даже если не может получить доступ к блокировке. Предыдущий пример трижды пытается получить доступ к блокировке файла *guestbook.txt*, ожидая пять секунд перед каждой попыткой.

Блокирование с помощью функции `flock()` работает не при всех условиях, примером тому служат некоторые реализации NFS. Кроме того, `flock()` не поддерживается в Windows 95, 98 или ME. Для эмулирования блокировки файла в таких случаях используется каталог в качестве индикатора монопольной блокировки. Это отдельный пустой каталог, наличие которого означает, что файл данных заблокирован. Перед открытием файла с данными создайте каталог блокировки, а по окончании работы с файлом данных удалите этот каталог. Как показано ниже, во всем остальном программа доступа к файлу выглядит так же:

```
$fh = fopen('guestbook.txt','a')         or die($php_errormsg);

// выполняем цикл, пока не создадим каталог блокировки
$locked = 0;
while (! $locked) {
    if (@mkdir('guestbook.txt.lock',0777)) {
        $locked = 1;
    } else {
        sleep(1);
    }
}

if (-1 == fwrite($fh,$_REQUEST['guestbook_entry'])) {
    rmdir('guestbook.txt.lock');
    die($php_errormsg);
}
if (! fclose($fh)) {
    rmdir('guestbook.txt.lock');
    die($php_errormsg);
}
rmdir('guestbook.txt.lock')              or die($php_errormsg);
```

Для индикации блокировки вместо файла используется каталог, поскольку функция `mkdir()` не в состоянии создать каталог, если он уже существует. Это дает возможность в одной операции проверить наличие индикатора и создать его, если он не существует. Однако эта «ловушка для ошибок» должна быть очищена перед выходом путем удаления каталога. Если каталог останется на месте, то в дальнейшем ни один процесс не сможет получить доступ к блокировке с помощью создания каталога.

Если в качестве индикатора выступает файл, то код для его создания выглядит так:

```
$locked = 0;
while (! $locked) {
    if (! file_exists('guestbook.txt.lock')) {
        touch('guestbook.txt.lock');
```

```
        $locked = 1;
    } else {
        sleep(1);
    }
}
```

При большом трафике из этого может ничего не получиться, поскольку сначала с помощью функции `file_exists()` проверяется существование блокировки, а затем с помощью функции `touch()` создается блокировка. После того как один процесс вызвал функцию `file_exists()`, другой процесс может вызвать функцию `touch()` до того, как ее вызовет первый процесс. В этом случае оба процесса будут считать, что они получили исключительный (монопольный) доступ к файлу, хотя ни один из них этого не сделал. В случае применения функции `mkdir()` нет паузы между проверкой существования и созданием, поэтому процессу, который создает каталог, гарантирован исключительный доступ.

См. также

Документацию по функции `flock()` на *http://www.php.net/flock*.

18.25. Чтение и запись сжатых файлов

Задача

Необходимо прочитать или записать сжатые файлы.

Решение

Для чтения или записи файлов *gzip* применяется расширение PHP *zlib*. Чтобы прочитать сжатый файл:

```
$zh = gzopen('file.gz','r') or die("can't open: $php_errormsg");
while ($line = gzgets($zh,1024)) {
    // переменная $line - это следующая строка несжатого файла,
    // вплоть до 1024 байт
}
gzclose($zh) or die("can't close: $php_errormsg");
```

Ниже показано, как записать сжатый файл:

```
$zh = gzopen('file.gz','w') or die("can't open: $php_errormsg");
if (-1 == gzwrite($zh,$s))   { die("can't write: $php_errormsg"); }
gzclose($zh)                 or die("can't close: $php_errormsg");
```

Обсуждение

Расширение *zlib* поддерживает версии многих функций доступа к файлам, таких как `fopen()`, `fread()` и `fwrite()` (называемых `gzopen()`, `gzread()`, `gzwrite()` и т. д.), которые прозрачно упаковывают данные при записи и распаковывают при чтении. Алгоритм сжатия, реализованный в *zlib*, совместим с утилитами *gzip* и *gunzip*.

Например, функция gzgets($zp,1024) работает подобно fgets($fh,1024). Она читает до 1023 байт, останавливаясь раньше, если встречает EOF или символ новой строки. Для функции gzgets() это означает 1023 несжатых байт.

Однако функция gzseek() работает иначе, чем функция fseek(). Она поддерживает только поиск определенного количества байт с начала файлового потока (аргумент SEEK_SET в функции fseek()). Поиск в прямом направлении (начиная с текущей позиции) поддерживается, только если файл открыт на запись (файл дополняется последовательностью сжатых нулей). Поиск в обратном направлении поддерживается в файле, открытом на чтение, но он очень медленный.

Расширение *zlib* также имеет несколько функций для создания сжатых строк. Функция gzencode() сжимает строку и обеспечивает ее корректными заголовками и форматированием для совместимости с *gunzip*. Приведем простую программу *gzip*:

```
$in_file = $_SERVER['argv'][1];
$out_file = $_SERVER['argv'][1].'.gz';

$ifh = fopen($in_file,'rb')  or die("can't open $in_file: $php_errormsg");
$ofh = fopen($out_file,'wb') or die("can't open $out_file: $php_errormsg");

$encoded = gzencode(fread($ifh,filesize($in_file)))
                             or die("can't encode data: $php_errormsg");

if (-1 == fwrite($ofh,$encoded)) { die("can't write: $php_errormsg"); }
fclose($ofh)                 or die("can't close $out_file: $php_errormsg");
fclose($ifh)                 or die("can't close $in_file: $php_errormsg");
```

Суть этой программы заключена в следующих строках:

```
$encoded = gzencode(fread($ifh,filesize($in_file)))
                             or die("can't encode data: $php_errormsg);
if (-1 == fwrite($ofh,$encoded)) { die("can't write: $php_errormsg"); }
```

Сжатое содержание переменной $in_file запоминается в переменной $encoded, а затем записывается в файл $out_file с помощью функции fwrite().

Функции gzencode() можно передать второй аргумент, задающий степень сжатия. Устанавливаем режим без сжатия, указав значение 0, и максимальное сжатие, указав значение 9. Уровень по умолчанию равен 1. Чтобы настроить простую программу *gzip* на максимальное сжатие, надо записать кодирующую строку так:

```
$encoded = gzencode(fread($ifh,filesize($in_file)),9)
                             or die("can't encode data: $php_errormsg);
```

Можно также упаковывать и распаковывать строки без *gzip*-совместимых заголовков с помощью gzcompress() и gzuncompress().

См. также

Рецепт 18.26 о программе, извлекающей файлы из ZIP-архива; документацию по расширению *zlib* на *http://www.php.net/zlib*; *zlib* можно загрузить с *http://www.gzip.org/ zlib/*; подробное описание алгоритма *zlib* в RFC 1950 (*http://www.faqs.org/rfcs/ rfc1950.html*) и RFC 1951 (*http://www.faqs.org/rfcs/rfc1951.html*).

18.26. Программа: Unzip

Программа *unzip.php*, показанная в примере 18.5, извлекает файлы из ZIP-архива. Она основана на функции `pc_mkdir_parents()`, которая определена в рецепте 19.10. Программа также требует установки PHP-расширения *zip*. Документацию по расширению *zip* можно найти на *http://www.php.net/zip*.

Эта программа принимает несколько аргументов командной строки. Первый – имя ZIP-архива, который следует распаковать. По умолчанию она распаковывает все файлы архива. Если в командной строке представлены дополнительные аргументы, то распаковываются только те файлы, имена которых совпадают с каким-либо из этих аргументов. Внутри ZIP-архива должен быть указан полный путь к файлу. Если файл *turtles.html* находится в ZIP-архиве внутри каталога *animals*, то для того чтобы распаковать файл, программе *unzip.php* надо передать *animals/turtles.html*, а не просто *turtles.html*.

Каталоги внутри ZIP-архивов хранятся как файлы, содержащие 0 байт, поэтому программа *unzip.php* не пытается их создавать. Вместо этого перед созданием какого-либо файла она вызывает функцию `pc_mkdir_parents()` для создания всех родительских каталогов файла, если это необходимо. Предположим, что программа *unzip.php* видит в ZIP-архиве следующие вхождения:

```
animals (0 bytes)
animals/frogs/ribbit.html (2123 bytes)
animals/turtles.html   (1232 bytes)
```

Она игнорирует *animals* как имеющий длину 0 байт. Затем она вызывает функцию `pc_mkdir_parents()` для *animals/frogs*, создавая и *animals*, и *animals/frogs*, и записывает файл *ribbit.html* в каталог *animals/frogs*. Каталог *animals* уже существует, поэтому когда программа доходит до *animals/turtles.html*, она записывает файл *turtles.html* без создания каких-либо дополнительных каталогов.

Пример 18.5. unzip.php

```
// первый аргумент - это zip-файл
$in_file = $_SERVER['argv'][1];

// любые другие аргументы - это указанные архивные файлы для распаковки
if ($_SERVER['argc'] > 2) {
```

```
        $all_files = 0;
        for ($i = 2; $i < $_SERVER['argc']; $i++) {
            $out_files[$_SERVER['argv'][$i]] = true;
        }
    } else {
        // если не указано других файлов, то распаковываем все файлы
        $all_files = true;
    }

    $z = zip_open($in_file) or die("can't open $in_file: $php_errormsg");
    while ($entry = zip_read($z)) {

        $entry_name = zip_entry_name($entry);

        // проверяем, должны ли распаковываться все файлы или находится
        // ли имя этого файла в списке файлов для распаковки
        if ($all_files || $out_files[$entry_name]) {

            // продолжаем, только если длина файла отлична от 0 байт
            if (zip_entry_filesize($entry)) {
                $dir = dirname($entry_name);

                // создаем все необходимые каталоги, указанные в пути к файлу
                if (! is_dir($dir)) { pc_mkdir_parents($dir); }

                $file = basename($entry_name);

                if (zip_entry_open($z,$entry)) {
                    if ($fh = fopen($dir.'/'.$file,'w')) {
                        // записываем весь файл
                        fwrite($fh,
                               zip_entry_read($entry,zip_entry_filesize($entry)))
                            or error_log("can't write: $php_errormsg");
                        fclose($fh) or error_log("can't close: $php_errormsg");
                    } else {
                        error_log("can't open $dir/$file: $php_errormsg");
                    }
                    zip_entry_close($entry);
                } else {
                    error_log("can't open entry $entry_name: $php_errormsg");
                }
            }
        }
    }
```

См. также

Рецепт 18.25 о чтении и записи файлов, сжатых *zlib*; рецепт 19.10 о функции `pc_mkdir_parents()`; документацию по расширению *zip* на *http://www.php.net/zip*.

19

Каталоги

19.0. Введение

Файловая система хранит не только фактическое содержимое файлов, но и массу дополнительной информации о них. В том числе такие подробности, как размер и права доступа к файлу, каталог, где он находится. При работе с файлами, возможно, потребуется манипулировать этими метаданными. PHP предоставляет множество функций для чтения и обработки каталогов, содержимого каталогов и атрибутов файлов. Как и другие разделы PHP, связанные с файлами, эти функции с некоторыми упрощениями подобны C-функциям, решающим те же самые задачи.

Файлы организуются с помощью *индексных узлов*, или *i-узлов* (*inodes*). Каждый файл (и другие части файловой системы, такие как каталоги, устройства и ссылки) имеет свой собственный i-узел, содержащий указатель на местоположение блоков данных файла, а также на всю метаинформацию об этом файле. Блоки данных каталога содержат имена файлов, хранящихся в этом каталоге, и i-узел каждого файла.

PHP предоставляет два способа просмотра каталога для определения содержащихся в нем файлов. Первый способ состоит в том, чтобы вызвать функцию `opendir()` для получения дескриптора каталога, функцию `readdir()` для выполнения цикла по файлам и функцию `closedir()` для закрытия дескриптора каталога:

```
$d = opendir('/usr/local/images') or die($php_errormsg);
while (false !== ($f = readdir($d))) {
    // обрабатываем файл
}
closedir($d);
```

Второй способ заключается в использовании класса каталога. Создаем экземпляр класса с помощью функции `dir()`, читаем каждое имя файла с помощью метода `read()` и закрываем каталог, вызвав функцию `close()`:

```
$d = dir('/usr/local/images') or die($php_errormsg);
while (false !== ($f = $d->read())) {
```

```
    // обрабатываем файл
}
$d->close();
```

В рецепте 19.7 показано, как при помощи функции `opendir()` или `dir()` обработать каждый файл в каталоге. Создание новых каталогов рассматривается в рецепте 19.10, а удалению каталогов посвящен рецепт 19.11.

Файловая система хранит не только файлы и каталоги. В UNIX она может также хранить символические ссылки. Существуют специальные файлы, в которых находится указатель на другой файл. Можно удалить эту ссылку, не оказывая никакого влияния на файл, на который она указывает. Для создания символической ссылки предназначена функция `symlink()`:

```
symlink('/usr/local/images','/www/docroot/images') or die($php_errormsg);
```

Так создается символическая ссылка с именем *images* в каталоге */www/docroot*, указывающая на */usr/local/images*.

Для отыскания информации о файле, каталоге или ссылке надо обследовать соответствующий индексный узел. Функция `stat()` извлекает метаданные из i-узла. Она обсуждается в рецепте 19.2. Многие функции PHP вызывают функцию `stat()` для предоставления определенной части информации о файле. Они перечислены в табл. 19.1.

Таблица 19.1. Функции, предоставляющие информацию о файле

Имя функции	Какую информацию о файле предоставляет функция?
`file_exists()`	Файл существует?
`fileatime()`	Время последнего доступа
`filectime()`	Время последнего изменения метаданных
`filegroup()`	Группа (числовое значение)
`fileinode()`	Номер i-узла
`filemtime()`	Время последнего изменения содержания
`fileowner()`	Владелец (числовое значение)
`fileperms()`	Права доступа (десятичное числовое значение)
`filesize()`	Размер
`filetype()`	Тип (fifo, char, dir, block, link, file, unknown)
`is_dir()`	Это каталог?
`is_executable()`	Это исполняемый файл?
`is_file()`	Это обычный файл?
`is_link()`	Это символическая ссылка?
`is_readable()`	Можно читать?
`is_writable()`	Можно записывать?

В UNIX права доступа к файлу определяют, какие операции могут выполнять над ним собственник файла, пользователи из той же группы и все пользователи. Под операциями понимаются чтение, запись и выполнение. Для программ право на выполнение означает право на запуск программы, для каталогов – это право поиска и просмотра файлов в нем.

Права доступа в UNIX могут также содержать бит смены идентификатора пользователя (бит setuid), бит смены идентификатора группы (бит setgid) и бит sticky, или «липкий» бит. Бит смены идентификатора пользователя означает, что программа всегда выполняется с идентификатором и правами своего владельца. Бит смены идентификатора группы означает, что программа всегда выполняется с идентификатором и правами своей группы. Для каталога бит смены идентификатора группы означает, что новые файлы в каталоге по умолчанию создаются от имени той же самой группы, что и каталог. Липкий бит удобен для каталогов, в которых пользователи совместно хранят файлы, поскольку он не дает тем, у кого есть права на запись в каталог, но кто не относится к суперпользователям, удалять файлы этого каталога, пока они не станут собственниками файла или каталога.

При установке прав доступа с помощью функции `chmod()` (см. рецепт 19.3) они должны быть представлены в виде восьмеричного числа. Это число состоит из четырех цифр. Первая цифра – это любая специальная установка для файла (например, setuid или setgid). Вторая – цифра – это права доступа пользователя, определяющие, что может делать владелец файла. Третья цифра – права доступа группы, определяющие, что могут делать пользователи, относящиеся к группе файла. Четвертая цифра – это права доступа всего мира; эти права определяют, что могут делать все остальные пользователи. Чтобы вычислить соответствующее значение для каждой цифры, сложите вместе желаемые права доступа, относящиеся к этой цифре, взяв значения из табл. 19.2. Например, значение прав доступа, равное `0644`, соответствует следующему: здесь нет специальных установок (`0`), владелец файла может читать и записывать файл (`6`, что равно `4` (чтение) + `2` (запись)), пользователи из группы файла могут читать файл (первая цифра `4`), а все остальные пользователи также могут читать файл (вторая цифра `4`). Значение прав доступа, равное `4644`, имеет такой же смысл, за исключением того, что для файла также установлен бит setuid.

Таблица 19.2. Значения прав доступа к файлу

Значение	Смысл права доступа	Смысл специальной установки
4	Чтение	setuid
2	Запись	setgid
1	Выполнение	липкий

На права доступа вновь созданных файлов и каталогов влияет установка, называемая *маской* (*umask*) и представляющая собой значение, ко-

торое вычитается (или маскирует) из начальных прав доступа файла (0666) или каталога (0777). Например, если маска равна 0022, то значение прав доступа по умолчанию для вновь созданного с помощью функции touch() или fopen() файла равно 0644, а права доступа по умолчанию для каталога, вновь созданного с помощью функции mkdir(), равны 0755. Получить текущее значение или установить маску можно посредством функции umask(). Она возвращает текущую маску и, если аргумент не соответствует ей, изменяет маску, присваивая ей значение этого аргумента. Например, ниже показано, как установить права доступа для вновь созданных файлов, не позволяющие получить доступ к ним никому, кроме владельца (и суперпользователя):

```
$old_umask = umask(0077);
touch('secret-file.txt');
umask($old_umask);
```

Первый вызов функции umask() маскирует все права доступа для группы и всего мира. После создания файла второй вызов функции umask() восстанавливает предыдущее значение маски. Если PHP запущен как серверный модуль, то он восстанавливает значение маски по умолчанию в конце каждого запроса. Подобно другим функциям, имеющим отношение к правам доступа, функция umask() не работает в Windows.

19.1. Получение и установка меток даты/времени файла

Задача

Необходимо узнать время последнего изменения или доступа к файлу либо обновить время доступа или изменения; например, вы хотите показать время последнего изменения каждой страницы вашего веб-сайта.

Решение

Функции fileatime(), filemtime() и filectime() возвращают время последнего доступа, время изменения файла и модификации его метаданных:

```
$last_access = fileatime('larry.php');
$last_modification = filemtime('moe.php');
$last_change = filectime('curly.php');
```

Функция touch() изменяет время модификации файла:

```
touch('shemp.php');             // устанавливаем время модификации,
                                // равным текущему времени
touch('joe.php',$timestamp);    // устанавливаем время модификации,
                                // равным $timestamp
```

Обсуждение

Функция `fileatime()` возвращает время последнего открытия файла на чтение или запись, функция `filemtime()` — время последнего изменения содержимого файла, а функция `filectime()` — время последнего изменения содержимого или метаданных файла (таких как владелец или права доступа). Каждая функция возвращает время в виде метки даты/времени UNIX.

Время модификации файла может быть обновлено с помощью функции `touch()`. Без второго аргумента функция `touch()` устанавливает время модификации, равным текущим дате и времени. Чтобы установить время модификации файла в определенное значение, передайте функции `touch()` в качестве второго аргумента это значение в виде метки даты/времени UNIX.

Следующий код выводит время последнего обновления страницы веб-сайта:

```
print "Last Modified: ".strftime('%c',filemtime
                                ($_SERVER['SCRIPT_FILENAME']));
```

См. также

Документацию по функции `fileatime()` на *http://www.php.net/fileatime*, по функции `filemtime()` на *http://www.php.net/filemtime* и по функции `filectime()` на *http://www.php.net/filectime*.

19.2. Получение информации о файле

Задача

Необходимо прочитать метаданные файла, например права доступа и имя владельца.

Решение

Вызовите функцию `stat()`, которая возвращает массив информации о файле:

```
$info = stat('harpo.php');
```

Обсуждение

Функция `stat()` возвращает массив информации о файле и с числовыми, и со строковыми индексами. Элементы массива представлены в табл. 19.3.

Таблица 19.3. Информация, возвращаемая функцией stat( )

Числовой индекс	Строковый индекс	Значение
0	dev	Устройство
1	ino	Inode
2	mode	Права доступа
3	nlink	Счетчик ссылок
4	uid	Идентификатор владельца-пользователя
5	gid	Идентификатор владельца-группы
6	rdev	Тип inode устройств (–1 в Windows)
7	size	Размер (в байтах)
8	atime	Время последнего доступа (метка даты/времени UNIX)
9	mtime	Время последнего изменения содержимого (метка даты/времени Unix)
10	ctime	Время последнего изменения содержимого или метаданных (метка даты/времени UNIX)
11	blksize	Размер блока ввода/вывода (–1 в Windows)
12	blocks	Количество блоков, расположенных в этом файле

Элемент mode возвращенного массива содержит права доступа в виде целого числа по основанию 10. Это сбивает с толку, поскольку права доступа обычно выражаются либо символически (т. е. вывод команды *ls* -rw-r--r--), либо в виде восьмеричного числа (т. е. 0644). Конвертировать права доступа в более понятный формат позволяет функция base_convert(), преобразующая права доступа в восьмеричное число:

```
$file_info = stat('/tmp/session.txt');
$permissions = base_convert($file_info['mode'],10,8);
```

В результате получим восьмеричное число из шести цифр. Например, если команда *ls* показывает о */tmp/session.txt* следующее:

```
-rw-rw-r--    1 sklar    sklar          12 Oct 23 17:55 /tmp/session.txt
```

то значение элемента $file_info['mode'] равно 33204, а значение переменной $permissions равно 100664. Последние три цифры (664) определяют права доступа к файлу: пользователя (чтение и запись), группы (чтение и запись) и всех остальных (чтение). Третья цифра, 0, означает, что для файла не установлен ни бит смены идентификатора пользователя (setuid), ни бит смены идентификатора группы (setgid). Крайнее слева число 10 означает, что это обычный файл (а не сокет, символическая ссылка или другой специальный файл).

Поскольку функция `stat()` возвращает массив как с числовым, так и со строковым индексами, то в результате выполнения цикла `foreach` по этому массиву мы получаем две копии каждого значения. И поэтому надо обратиться к циклу `for` от элемента 0 до элемента 12 возвращенного массива.

Вызов функции `stat()` для символической ссылки возвращает информацию о файле, на который указывает эта символическая ссылка. Для получения информации о самой символической ссылке применяется функция `lstat()`.

Функция `fstat()` аналогична рассмотренной `stat()` и принимает дескриптор файла (возвращенный функцией `fopen()` или `popen()`) в качестве аргумента. Функцию `fstat()` можно применять только для локальных файлов и нельзя – к URL, передаваемым функции `fopen()`.

PHP-функция `stat()` осуществляет дорогостоящий базовый системный вызов *stat(2)*. Для минимизации накладных расходов PHP кэширует результаты вызова *stat(2)*. Поэтому если вызвать функцию `stat()` для файла, изменить его права и снова вызвать функцию `stat()` для этого же файла, то будут получены те же самые результаты. Чтобы заставить PHP перегрузить метаданные файла, вызовите функцию `clearstatcache()`, которая сбросит информацию буфера PHP. Этот же кэш нужен PHP и для других функций, возвращающих метаданные: `file_exists()`, `fileatime()`, `filectime()`, `filegroup()`, `fileinode()`, `filemtime()`, `fileowner()`, `fileperms()`, `filesize()`, `filetype()`, `fstat()`, `is_dir()`, `is_executable()`, `is_file()`, `is_link()`, `is_readable()`, `is_writable()` и `lstat()`.

См. также

Документацию по функции `stat()` на *http://www.php.net/stat*, по функции `lstat()` на *http://www.php.net/lstat*, по функции `fstat()` на *http://www.php.net/fstat* и по функции `clearstatcache()` на *http://www.php.net/clearstatcache*.

19.3. Изменение прав доступа к файлу или его владельца

Задача

Необходимо изменить права доступа к файлу или его владельца. Например, надо не дать возможности другим пользователям увидеть закрытые данные, хранящиеся в файле.

Решение

Для изменения прав доступа к файлу применяется функция `chmod()`:

```
chmod('/home/user/secrets.txt',0400);
```

Функция chown() позволяет изменить владельца файла, а функция chgrp() — изменить группу файла:

```
chown('/tmp/myfile.txt','sklar');           // указываем пользователя
                                               по имени
chgrp('/home/sklar/schedule.txt','soccer'); // указываем группу по имени

chown('/tmp/myfile.txt',5001);              // указываем пользователя
                                               по его идентификатору
chgrp('/home/sklar/schedule.txt',102);      // указываем группу
                                               по ее идентификатору
```

Обсуждение

Права доступа, передаваемые функции chmod(), должны быть указаны в виде восьмеричного числа.

Суперпользователь может изменить права доступа, владельца и группу для любого файла. Для других пользователей имеются ограничения. Они могут только изменять права доступа и группу файлов, которыми владеют, но не могут менять владельца файлов. Несуперпользователи могут также изменять группу файла только на одну из групп, к которым они принадлежат сами.

Функции chmod(), chgrp() и chown() не работают в Windows.

См. также

Документацию по функции chmod() на *http://www.php.net/chmod*, по функции chown() на *http://www.php.net/chown* и по функции chgrp() на *http://www.php.net/chgrp*.

19.4. Разделение имени файла на составляющие

Задача

Необходимо определить путь к файлу и его имя; например, требуется создать файл в том же каталоге, в котором находится существующий.

Решение

Для выделения имени файла применяется функция basename(), а функция dirname() — для выделения пути к нему:

```
$full_name = '/usr/local/php/php.ini';
$base = basename($full_name);  // значение переменной $base равно php.ini
$dir  = dirname($full_name);   // значение переменной $dir
                                  равно /usr/local/php
```

Функция pathinfo() позволяет получить из ассоциативного массива имя каталога, базовое имя и расширение:

```
$info = pathinfo('/usr/local/php/php.ini');
```

Обсуждение

Для создания временного файла в каталоге существующего файла вызовите функцию dirname(), чтобы найти каталог, и передайте его функции tempnam():

```
$dir = dirname($existing_file);
$temp = tempnam($dir,'temp');
$temp_fh = fopen($temp,'w');
```

Элементы ассоциативного массива, возвращенные функцией pathinfo(), — это dirname, basename и extension:

```
$info = pathinfo('/usr/local/php/php.ini');
print_r($info);
Array
(
    [dirname] => /usr/local/php
    [basename] => php.ini
    [extension] => ini
)
```

Можно также передать функции basename() необязательный суффикс, чтобы удалить его из имени файла. В этом случае переменная $base получит значение *php*:

```
$base = basename('/usr/local/php/php.ini','.ini');
```

Применение функций basename(), dirname() и pathinfo() более переносимо, чем просто разделение полного имени файла символом /, поскольку в них используется разделитель, соответствующий операционной системе. В Windows эти функции считают разделителем каталогов и /, и \. На других платформах используется только символ /.

В PHP нет встроенной функции для обратного объединения в полное имя файла частей, возвращенных функциями basename(), dirname() и pathinfo(). Чтобы сделать это, надо объединить части с помощью символов . и /:

```
$dirname = '/usr/local/php';
$basename = 'php';
$extension = 'ini';

$full_name = $dirname . '/' . $basename . '.' . $extension;
```

Можно спокойно передать полученное таким образом полное имя файла файловой функции PHP в Windows, поскольку PHP в Windows принимает символ / в качестве разделителя каталогов.

См. также

Документацию по функции basename() на *http://www.php.net/basename*, по функции dirname() на *http://www.php.net/dirname* и по функции pathinfo() на *http://www.php.net/pathinfo*.

19.5. Удаление файла

Задача

Необходимо удалить файл.

Решение

Это делается при помощи функции `unlink()`:

```
unlink($file) or die ("can't delete $file: $php_errormsg");
```

Обсуждение

Функция `unlink()` может удалять только те файлы, которые может удалить пользователь процесса PHP. Если функция `unlink()` работает не так, как положено, то следует проверить права доступа к файлу и то, как запускается PHP.

См. также

Документацию по функции `unlink()` на *http://www.php.net/unlink*.

19.6. Копирование и перемещение файла

Задача

Необходимо скопировать или переместить файл.

Решение

Для копирования файла применяется функция `copy()`:

```
copy($old,$new) or die("couldn't copy $old to $new: $php_errormsg");
```

А функция `rename()` позволяет перемещать файл:

```
rename($old,$new) or die("couldn't move $old to $new: $php_errormsg");
```

Обсуждение

В UNIX функция `rename()` не может перемещать файл по файловой системе. Чтобы это сделать, скопируйте файл в новое место и удалите старый файл:

```
if (copy("/tmp/code.c","/usr/local/src/code.c")) {
  unlink("/tmp/code.c");
}
```

Для выполнения множественного копирования или перемещения вызывайте функцию `copy()` или `rename()` в цикле. Каждая функция во время текущего вызова может работать только с одним файлом.

См. также

Документацию по функции `copy()` на *http://www.php.net/copy* и по функции `rename()` на *http://www.php.net/rename*.

19.7. Обработка всех файлов в каталоге

Необходимо выполнить цикл по всем файлам в каталоге. Например, требуется создать элемент `select` в форме, содержащей список всех файлов в каталоге.

Решение

Получите дескриптор каталога с помощью функции `opendir()`, а затем извлеките имя каждого файла, вызвав функцию `readdir()`:

```
$d = opendir('/tmp') or die($php_errormsg);
while (false !== ($f = readdir($d))) {
    print "$f\n";
}
closedir($d);
```

Обсуждение

Фрагмент кода в разделе «Решение» проверяет возвращенное функцией `readdir()` значение с помощью оператора тождественного неравенства (`!==`), поэтому этот код работает с именами файлов, которые могут программно интерпретироваться как `false`, например, если имя файла *0*.

Функция `readdir()` возвращает любой элемент каталога – является ли он файлом, каталогом или чем-нибудь еще (например, ссылкой или сокетом). Сюда же входят метаэлементы «.» (текущий каталог) и «..» (родительский каталог). Для того чтобы вернуть только файлы, вызовите также функцию `is_file()`:

```
print '<select name="files">';
$d = opendir('/usr/local/upload') or die($php_errormsg);
while (false !== ($f = readdir($d))) {
    if (is_file("/usr/local/upload/$f")) {
        print '<option> ' . $f . '</option>';
    }
}
closedir($d);
print '</select>';
```

Функция `readdir()` возвращает только имя файла каждого вхождения каталога, а не полное имя пути, поэтому необходимо предварять именем каталога переменную `$f`, прежде чем передавать ее функции `is_file()`.

В PHP есть также объектно-ориентированный интерфейс к информации каталога. Функция `dir()` возвращает объект, в котором можно вызвать методы `read()`, `rewind()` и `close()`, действующие подобно функци-

ям readdir(), rewinddir() и closedir(). У него есть также свойство $path, содержащее полное имя пути к открытому каталогу.

Ниже показано, как выполнить цикл по файлам с помощью объектно-ориентированного интерфейса:

```
print '<select name="files">';
$d = dir('/usr/local/upload') or die($php_errormsg);
while (false !== ($f = $d->read())) {
    if (is_file($d->path.'/'.$f)) {
        print '<option> ' . $f . '</option>';
    }
}
$d->close();
```

В этом примере $d->path равно */usr/local/upload*.

См. также

Документацию по функции opendir() на *http://www.php.net/opendir*, по функции readdir() на *http://www.php.net/readdir* и по классу каталога на *http://www.php.net/class.dir*.

19.8. Получение списка имен файлов, соответствующих шаблону

Задача

Необходимо найти все имена файлов, соответствующие шаблону.

Решение

Если шаблон представляет собой регулярное выражение, то прочитайте каждый файл из каталога и проверьте его имя с помощью функции preg_match():

```
$d = dir('/tmp') or die($php_errormsg);
while (false !== ($f = $d->read())) {
    // соответствует только именам из букв
    if (preg_match('/^[a-zA-Z]+$/',$f)) {
        print "$f\n";
    }
}
$d->close();
```

Обсуждение

Если шаблон представляет собой универсальный символ оболочки (*.*), используйте оператор обратного апострофа в командах *ls* (UNIX) или *dir* (Windows) для получения соответствующих имен файлов. Для UNIX:

```
$files = explode("\n",`ls -1 *.gif`);
```

```
foreach ($files as $file) {
  print "$b\n";
}
```

Для Windows:

```
$files = explode("\n",`dir /b *.gif`);
foreach ($files as $file) {
  print "$b\n";
}
```

См. также

Подробную информацию о выполнении цикла по каждому файлу каталога в рецепте 19.7; информацию о сравнении с помощью шаблона оболочки на *http://www.gnu.org/manual/bash/html_node/bashref_35.html*.

19.9. Обработка всех файлов в каталоге

Задача

Необходимо что-нибудь сделать со всеми файлами в каталоге и в любых его подкаталогах.

Решение

Функция `pc_process_dir()`, показанная в примере 19.1, возвращает список всех файлов в данном каталоге и ниже.

Пример 19.1. pc_process_dir( )

```
function pc_process_dir($dir_name,$max_depth = 10,$depth = 0) {
    if ($depth >= $max_depth) {
        error_log("Reached max depth $max_depth in $dir_name.");
        return false;
    }
    $subdirectories = array();
    $files = array();
    if (is_dir($dir_name) && is_readable($dir_name)) {
        $d = dir($dir_name);
        while (false !== ($f = $d->read())) {
            // пропускаем . и ..
            if (('.' == $f) || ('..' == $f)) {
                continue;
            }
            if (is_dir("$dir_name/$f")) {
                array_push($subdirectories,"$dir_name/$f");
            } else {
                array_push($files,"$dir_name/$f");
            }
        }
        $d->close();
        foreach ($subdirectories as $subdirectory) {
```

```
            $files = array_merge($files,pc_process_dir($subdirectory,
                                 $max_depth,$depth+1));
        }
    }
    return $files;
}
```

Обсуждение

Вот пример: если каталог */tmp* содержит файлы *a* и *b*, а также каталог *c*, а каталог */tmp/c* содержит файлы *d* и *e*, то функция `pc_process_dir('/tmp')` возвращает массив с элементами */tmp/ a*, */tmp/b*, */tmp/c/d* и */tmp/c/e*. Чтобы выполнить операцию над каждым файлом, организуйте цикл по массиву:

```
$files = pc_process_dir('/tmp');
foreach ($files as $file) {
  print "$file was last accessed at ".strftime('%c',fileatime($file)).".\n";
}
```

Можно не возвращать массив файлов, а написать функцию, обрабатывающую их по мере нахождения. Функция `pc_process_dir2()`, показанная в примере 19.2, делает это, принимая дополнительный аргумент – имя функции, вызываемой для каждого найденного файла.

Пример 19.2. pc_process_dir2( )

```
function pc_process_dir2($dir_name,$func_name,$max_depth = 10,$depth = 0) {
    if ($depth >= $max_depth) {
        error_log("Reached max depth $max_depth in $dir_name.");
        return false;
    }
    $subdirectories = array();
    $files = array();
    if (is_dir($dir_name) && is_readable($dir_name)) {
        $d = dir($dir_name);
        while (false !== ($f = $d->read())) {
            // пропускаем . и ..
            if (('.' == $f) || ('..' == $f)) {
                continue;
            }
            if (is_dir("$dir_name/$f")) {
                array_push($subdirectories,"$dir_name/$f");
            } else {
                $func_name("$dir_name/$f");
            }
        }
        $d->close();
        foreach ($subdirectories as $subdirectory) {
            pc_process_dir2($subdirectory,$func_name,$max_depth,$depth+1);
        }
    }
}
```

Функция pc_process_dir2() не возвращает список каталогов; вместо этого функция $func_name вызывается с файлом в качестве аргумента. Ниже показано, как вывести время последнего доступа:

```
function printatime($file) {
    print "$file was last accessed at ".strftime('%c',fileatime($file)).'\n';
}

pc_process_dir2('/tmp','printatime');
```

Эти две функции получают один и тот же результат, но вторая требует меньше памяти, поскольку не передаются потенциально большие массивы.

В функциях pc_process_dir() и pc_process_dir2() реализован алгоритм поиска типа «сначала вширь». При выполнении поиска такого типа функции обрабатывают все файлы в текущем каталоге, а затем последовательно переходят в каждый подкаталог. При выполнении поиска типа «сначала вглубь» они заходят в подкаталог сразу же, как только его обнаруживают, независимо от того, остались еще файлы в текущем каталоге или нет. Поиск типа «сначала вширь» более эффективен с точки зрения расходования памяти – перед тем как функция перейдет в подкаталоги, каждый указатель на текущий каталог закрывается (с помощью $d->close()), поэтому одновременно открыт только один указатель на каталог.

Функция is_dir() возвращает true, когда передана символическая ссылка на каталог, поэтому обе версии функций следуют символическим ссылкам во время путешествия вниз по дереву каталогов. Если вы не хотите следовать ссылкам, замените строку:

```
if (is_dir("$dir_name/$f")) {
```

на:

```
if (is_dir("$dir_name/$f") && (! is_link("$dir_name/$f"))) {
```

См. также

Рецепт 6.9, в котором обсуждаются переменные функции; документацию по функции is_dir() на *http://www.php.net/is-dir* и по функции is_link() на *http://www.php.net/is-link*.

19.10. Создание новых каталогов

Задача

Необходимо создать каталог.

Решение

Это делается при помощи функции mkdir():

```
mkdir('/tmp/apples',0777) or die($php_errormsg);
```

Обсуждение

Второй аргумент функции `mkdir()` – это режим доступа к новому каталогу, который должен быть восьмеричным числом. Текущая маска (umask) вычитается из этого значения прав доступа для создания прав доступа к новому каталогу. Поэтому если текущая маска равна `0002`, вызов `mkdir('/tmp/apples',0777)` устанавливает права доступа к результирующему каталогу, равные `0775` (пользователь и группа могут читать, писать и выполнять; остальные могут только читать и выполнять).

Встроенная в PHP функция `mkdir()` может создать новый каталог, только если существует родительский каталог. Например, если */tmp/a* не существует, то нельзя создать каталог */tmp/a/b*, пока не будет создан каталог */tmp/a*. Для создания каталога и его родителя существует два способа: можно вызвать системную программу `mkdir` или вызвать функцию `pc_mkdir_parents()`, показанную в примере 19.3. Чтобы использовать системную программу *mkdir* в UNIX, выполните:

```
system('/bin/mkdir -p '.escapeshellarg($directory));
```

В Windows:

```
system('mkdir '.escapeshellarg($directory));
```

Можно также вызвать функцию `pc_mkdir_parents()`, показанную в примере 19.3.

Пример 19.3. pc_mkdir_parents( )

```
function pc_mkdir_parents($d,$umask = 0777) {
    $dirs = array($d);
    $d = dirname($d);
    $last_dirname = '';
    while($last_dirname != $d) {
        array_unshift($dirs,$d);
        $last_dirname = $d;
        $d = dirname($d);
    }

    foreach ($dirs as $dir) {
        if (! file_exists($dir)) {
            if (! mkdir($dir,$umask)) {
                error_log("Can't make directory: $dir");
                return false;
            }
        } elseif (! is_dir($dir)) {
            error_log("$dir is not a directory");
            return false;
        }
    }
    return true;
}
```

Например:

```
pc_mkdir_parents('/usr/local/upload/test',0777);
```

См. также

Документацию по функции `mkdir()` на *http://www.php.net/mkdir*; системную документацию по программе *mkdir*, например справочную страницу *mkdir(1)* программы *man* в UNIX или текст справки *mkdir /?* в Windows.

19.11. Удаление каталога и его содержимого

Задача

Необходимо удалить каталог и все его содержимое, включая подкаталоги и их содержимое.

Решение

В UNIX надо выполнить команду *rm*:

```
$directory = escapeshellarg($directory);
exec("rm -rf $directory");
```

В Windows – команду *rmdir*:

```
$directory = escapeshellarg($directory);
exec("rmdir /s /q $directory");
```

Обсуждение

Очевидно, что удаление файлов может быть опасным. Не забудьте вызвать функцию `escapeshellarg()` для переменной `$directory`, чтобы не удалить непредусмотренные файлы.

Поскольку встроенная в PHP функция `rmdir()`, которая удаляет каталоги, работает только с пустыми каталогами, а функция `unlink()` не принимает групповые символы оболочки, то вызов системной программы намного легче, чем рекурсивное выполнение цикла по всем файлам в каталоге с их удалением и последующим удалением каждого каталога. Однако если внешняя утилита недоступна, то для удаления каждого подкаталога можно модифицировать функцию из рецепта 19.9.

См. также

Документацию по функции `rmdir()` на *http://www.php.net/rmdir*; системную документацию по *rm* или *rmdir*, например страницу помощи *rm(1)* программы man в UNIX или текст помощи *rmdir /?* в Windows.

19.12. Программа: Перечень каталогов веб-сервера

Программа *web-ls.php*, показанная в примере 19.4, выводит сведения о файлах корневого каталога документов веб-сервера, отформатированные аналогично выводу UNIX-команды *ls*. Ссылки на имена файлов выполнены так, чтобы можно было загрузить каждый файл, а ссылки на имена каталогов позволяют просматривать каждый каталог, как показано на рис. 19.1.

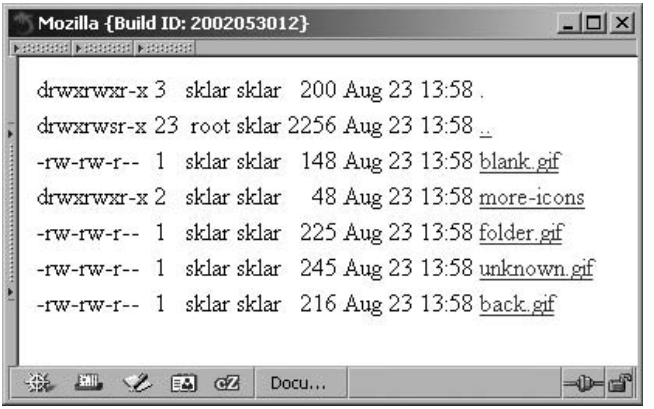

Рис. 19.1. *Веб-перечень*

Большинство строк предназначены для построения легкого для чтения представления прав доступа к файлу, но суть программы в цикле `while` в ее конце. Метод `$d->read()` дает имя каждого файла в каталоге. Затем функция `lstat()` извлекает информацию, а функция `printf()` выводит отформатированную информацию об этом файле.

Функция `mode_string()` и константы, которые она использует, преобразуют восьмеричное представление режима файла (т. е. `35316`) в более легкую для чтения строку (т. е. `-rwsrw-r--`).

Пример 19.4. web-ls.php

```
/* Битовые маски для определения прав доступа к файлу и типа.
 * Имена и значения, перечисленные ниже, POSIX-совместимы, отдельные системы
 * могут иметь свои собственные расширения.
 */

define('S_IFMT',0170000);   // маска для всех типов
define('S_IFSOCK',0140000); // тип: сокет
define('S_IFLNK',0120000);  // тип: символическая ссылка
define('S_IFREG',0100000);  // тип: обычный файл
define('S_IFBLK',0060000);  // тип: блочное устройство
define('S_IFDIR',0040000);  // тип: каталог
```

```
define('S_IFCHR',0020000);  // тип: символьное устройство
define('S_IFIFO',0010000);  // тип: fifo
define('S_ISUID',0004000);  // бит смены идентификатора
                               пользователя (set-uid)
define('S_ISGID',0002000);  // бит смены идентификатора группы (set-gid)
define('S_ISVTX',0001000);  // sticky-бит (липкий бит)
define('S_IRWXU',00700);    // маска для прав доступа владельца
define('S_IRUSR',00400);    // владелец: право на чтение
define('S_IWUSR',00200);    // владелец: право на запись
define('S_IXUSR',00100);    // владелец: право на выполнение
define('S_IRWXG',00070);    // маска для прав доступа группы
define('S_IRGRP',00040);    // группа: право на чтение
define('S_IWGRP',00020);    // группа: право на запись
define('S_IXGRP',00010);    // группа: право на выполнение
define('S_IRWXO',00007);    // маска для прав доступа остальных
define('S_IROTH',00004);    // остальные: право на чтение
define('S_IWOTH',00002);    // остальные: право на запись
define('S_IXOTH',00001);    // остальные: право на выполнение

/* mode_string() - это вспомогательная функция, которая принимает
 * восьмеричный режим и возвращает десятисимвольную строку, представляющую
 * тип файла и права доступа, которые соответствуют восьмеричному режиму.
 * Это PHP-версия функции mode_string() в пакете файловых утилит GNU.
 */
function mode_string($mode) {
  $s = array();

 // set type letter
  if (($mode & S_IFMT) == S_IFBLK) {
    $s[0] = 'b';
  } elseif (($mode & S_IFMT) == S_IFCHR) {
    $s[0] = 'c';
  } elseif (($mode & S_IFMT) == S_IFDIR) {
    $s[0] = 'd';
  } elseif (($mode & S_IFMT) ==  S_IFREG) {
    $s[0] = '-';
  } elseif (($mode & S_IFMT) ==  S_IFIFO) {
    $s[0] = 'p';
  } elseif (($mode & S_IFMT) == S_IFLNK) {
    $s[0] = 'l';
  } elseif (($mode & S_IFMT) == S_IFSOCK) {
    $s[0] = 's';
  }

  // устанавливаем права доступа пользователя
  $s[1] = $mode & S_IRUSR ? 'r' : '-';
  $s[2] = $mode & S_IWUSR ? 'w' : '-';
  $s[3] = $mode & S_IXUSR ? 'x' : '-';

  // устанавливаем права доступа группы
  $s[4] = $mode & S_IRGRP ? 'r' : '-';
  $s[5] = $mode & S_IWGRP ? 'w' : '-';
  $s[6] = $mode & S_IXGRP ? 'x' : '-';
```

```
  // устанавливаем права доступа остальных
  $s[7] = $mode & S_IROTH ? 'r' : '-';
  $s[8] = $mode & S_IWOTH ? 'w' : '-';
  $s[9] = $mode & S_IXOTH ? 'x' : '-';

  // устанавливаем символы выполнения для set-uid, set-gid и sticky
  if ($mode & S_ISUID) {
    if ($s[3] != 'x') {
      // set-uid, но не исполняемый владельцем
      $s[3] = 'S';
    } else {
      $s[3] = 's';
    }
  }

  if ($mode & S_ISGID) {
    if ($s[6] != 'x') {
      // set-gid, но не исполняемый группой
      $s[6] = 'S';
    } else {
      $s[6] = 's';
    }
  }

  if ($mode & S_ISVTX) {
    if ($s[9] != 'x') {
      // sticky, но не исполняемый остальными
      $s[9] = 'T';
    } else {
      $s[9] = 't';
    }
  }

  // возвращаем отформатированную строку
  return join('',$s);

}

// Начинаем в корневом документальном каталоге, если ничего не указано
if (isset($_REQUEST['dir'])) {
    $dir = $_REQUEST['dir'];
} else {
    $dir = '';
}

// находим $dir в файловой системе
$real_dir = realpath($_SERVER['DOCUMENT_ROOT'].$dir);

// проверяем, что $real_dir находится в документальном корневом каталоге
if (! preg_match('/^'.preg_quote($_SERVER['DOCUMENT_ROOT'],'/').'/',
                 $real_dir)) {
    die("$dir is not inside the document root");
}

// канонизируем $dir, удаляя из его начала корневой документальный каталог
```

```
$dir = substr_replace($real_dir,'',0,strlen($_SERVER['DOCUMENT_ROOT']));

// мы открываем каталог?
if (! is_dir($real_dir)) {
    die("$real_dir is not a directory");
}

// открываем указанный каталог
$d = dir($real_dir) or die("can't open $real_dir: $php_errormsg");

print '<table>';

// читаем каждое вхождение в каталоге
while (false !== ($f = $d->read())) {

    // получаем информацию об этом файле
    $s = lstat($d->path.'/'.$f);

    // переводим идентификатор пользователя в имя пользователя
    $user_info = posix_getpwuid($s['uid']);

    // переводим идентификатор группы в имя группы
    $group_info = posix_getgrgid($s['gid']);

    // форматируем данные для улучшения читаемости
    $date = strftime('%b %e %H:%M',$s['mtime']);

    // переводим восьмеричный режим в читаемую строку
    $mode = mode_string($s['mode']);

    $mode_type = substr($mode,0,1);
    if (($mode_type == 'c') || ($mode_type == 'b')) {
        /* если это блочное или символьное устройство, то выводим старший
         * и младший тип устройства вместо размера файла */
        $major = ($s['rdev'] >> 8) & 0xff;
        $minor = $s['rdev'] & 0xff;
        $size = sprintf('%3u, %3u',$major,$minor);
    } else {
        $size = $s['size'];
    }

    // форматируем <a href=""> вокруг имени файла
    // нет ссылки на текущий каталог
    if ('.' == $f) {
        $href = $f;
    } else {
        // не включаем «..» в ссылку на родительский каталог
        if ('..' == $f) {
            $href = urlencode(dirname($dir));
        } else {
            $href = urlencode($dir) . '/' . urlencode($f);
        }

        /* все, за исключением "/", должно быть в кодировке URL */
        $href = str_replace('%2F','/',$href);

        // просматриваем другие каталоги с помощью веб-команды ls
```

```
        if (is_dir(realpath($d->path . '/' . $f))) {
            $href = sprintf('<a href="%s?dir=%s">%s</a>',
                            $_SERVER['PHP_SELF'],$href,$f);
        } else {
            // ссылка на файлы для их загрузки
            $href= sprintf('<a href="%s">%s</a>',$href,$f);
        }

        // если это ссылка, то показываем также цель
        if ('l' == $mode_type) {
            $href .= ' -&gt; ' . readlink($d->path.'/'.$f);
        }
    }

    // выводим соответствующую информацию об этом файле
    printf('<tr><td>%s</td><td>%3u</td><td align="right">%s</td>
            <td align="right">%s</td><td align="right">%s</td>
            <td align="right">%s</td><td>%s</td></tr>',
           $mode,                // форматированная строка режима
           $s['nlink'],          // количество ссылок на этот файл
           $user_info['name'],   // имя пользователя-владельца
           $group_info['name'],  // имя группы
           $size,                // размер файла (или номера устройства)
           $date,                // дата и время последнего изменения
           $href);               // ссылка для просмотра или загрузки
}

print '</table>';
```

19.13. Программа: Поиск сайта

Программу *site-search.php*, показанную в примере 19.5, можно использовать как поисковую машину для сайтов от малого до среднего размеров, основанных на файлах.

Программа ищет условия поиска (в `$_REQUEST['term']`) во всех файлах в пределах определенного набора каталогов в корневом каталоге документов. Эти каталоги определяются в `$search_dirs`. Программа также последовательно заходит в подкаталоги и следует символическим ссылкам, но запоминает, какие файлы и каталоги она просмотрела, так что она не зацикливается.

Если найдены какие-либо страницы, содержащие условия поиска, то она печатает список ссылок на эти страницы, расположенные в соответствии с алфавитным порядком заголовков этих страниц. Если у страницы нет заголовка (между тегами `<title>` и `</title>`), то используется URI страницы относительно корневого документального каталога.

Программа ищет условия поиска между тегами `<body>` и `</body>` в каждом файле. Если на вашей странице между тегами `<body>` находится большой объем текста, который вы хотите исключить из поиска, то окружите текст, который следует искать, специальными HTML-коммен-

тариями, а затем модифицируйте $body_regex так, чтобы искать эти теги. Например, ваша страница выглядит следующим образом:

```
<body>

// Некоторые HTML-элементы для меню, заголовков и т. д.

<!-- search-start -->

<h1>Aliens Invade Earth</h1>

<h3>by H.G. Wells</h3>

<p>Aliens invaded earth today. Uh Oh.</p>

// Продолжение повествования

<!-- search-end -->

// Некоторые HTML-элементы для нижних колонтитулов и т. д.

</body>
```

Чтобы сравнивать условия поиска только с названием, автором и с изложением внутри HTML-комментариев, измените $body_regex на:

```
$body_regex = '#<!-- search-start -->(.*' .
preg_quote($_REQUEST['term'],'#').
              '.*)<!-- search-end -->#Sis';
```

Если вы не хотите, чтобы условия поиска сравнивались с текстом внутри тегов HTML или PHP на вашей странице, то добавьте вызов функции strip_tags() в программу, загружающую содержимое файла для поиска:

```
// загружаем содержимое файла в переменную $file
$file = strip_tags(join('',file($path)));
```

Пример 19.5. site-search.php

```
function pc_search_dir($dir) {
    global $body_regex,$title_regex,$seen;

    // массив для хранения соответствующих страниц
    $pages = array();

    // массив для хранения каталогов поиска
    $dirs = array();

    // отмечаем этот каталог как просмотренный,
    // чтобы не просматривать его снова
    $seen[realpath($dir)] = true;

    // если мы можем получить дескриптор этого каталога
    if (is_readable($dir) && ($d = dir($dir))) {
        // берем каждое имя файла в каталоге
        while (false !== ($f = $d->read())) {
            // строим полный путь к файлу
            $path = $d->path.'/'.$f;
```

```
        // если это обычный файл и мы можем прочитать его
        if (is_file($path) && is_readable($path)) {

            $realpath = realpath($path);
            // если мы уже видели этот файл,
            if ($seen[$realpath]) {
                // то пропускаем его
                continue;
            } else {
                  // в противном случае отмечаем его как просмотренный,
                // чтобы пропустить его, если придем к нему снова
                $seen[$realpath] = true;
            }

            // загружаем содержимое файла в переменную $file
            $file = join('',file($path));

            // если условия поиска находятся внутри
            // ограничителей body
            if (preg_match($body_regex,$file)) {

                // конструируем относительный URI файла, удаляя
                // документальный корневой каталог из полного пути
                $uri = substr_replace($path,'',0,strlen
                                    ($_SERVER['DOCUMENT_ROOT']));

                // Если страница имеет название, то находим его
               if (preg_match('#<title>(.*?)</title>#Sis',$file,$match)) {
                    // и добавляем название и URI в переменную $pages
                    array_push($pages,array($uri,$match[1]));
                } else {
                    // в противном случае используем URI
                    // в качестве названия
                    array_push($pages,array($uri,$uri));
                }
            }
        } else {
            // если элемент каталога - это допустимый подкаталог
            if (is_dir($path) && ('.' != $f) && ('..' != $f)) {
                // добавляем его к списку каталогов для просмотра
                array_push($dirs,$path);
            }
        }
    }
    $d->close();
}

/* Просматриваем каждый файл в каждом подкаталоге данного каталога
 * и добавляем найденные страницы из этих каталогов
 * к переменной $pages. Просматриваем только подкаталог,
 * который мы еще не видели.
*/
foreach ($dirs as $subdir) {
    $realdir = realpath($subdir);
```

```
        if (! $seen[$realdir]) {
            $seen[$realdir] = true;
            $pages = array_merge($pages,pc_search_dir($subdir));
        }
    }

    return $pages;
}

// вспомогательная функция для сортировки найденных страниц
// в алфавитном порядке их названий
function pc_page_sort($a,$b) {
    if ($a[1] == $b[1]) {
        return strcmp($a[0],$b[0]);
    } else {
        return ($a[1] > $b[1]);
    }
}

// массив для хранения страниц, соответствующих условиям поиска
$matching_pages = array();
// массив для хранения страниц во время сканирования условий поиска
$seen = array();
// каталоги для поиска ниже корневого документального каталога
$search_dirs = array('sports','movies','food');
// Регулярное выражение для использования в искомых файлах. Модификатор
// шаблона «S» говорит машине PCRE «изучать» regex для большей эффективности.
$body_regex = '#<body>(.*' . preg_quote($_REQUEST['term'],'#').
              '.*)</body>#Sis';

// добавляем соответствующие файлы, найденные в каждом каталоге,
// в $matching_pages
foreach ($search_dirs as $dir) {
    $matching_pages = array_merge($matching_pages,
                          pc_search_dir($_SERVER['DOCUMENT_ROOT'].'/'.$dir));
}

if (count($matching_pages)) {
    // сортируем найденные страницы по названию
    usort($matching_pages,'pc_page_sort');
    print '<ul>';
    // выводим каждое название со ссылкой на страницу
    foreach ($matching_pages as $k => $v) {
        print sprintf('<li> <a href="%s">%s</a>',$v[0],$v[1]);
    }
    print '</ul>';
} else {
    print 'No pages found.';
}
```

20

PHP на стороне клиента

20.0. Введение

PHP был создан для веб-программирования и до сих пор, главным образом, он используется с этой целью. Тем не менее последние версии PHP все в большей степени приобретают черты, характерные для языков сценариев общего назначения. Применение PHP для разработки сценариев, запускаемых из командной строки, особенно полезно, когда такие сценарии работают совместно с веб-приложениями. Если на сайте есть форум, то может появиться и желание каждые пять минут (или часов) запускать программу, сканирующую новые сообщения и информирующую вас о сообщениях, содержащих определенные ключевые слова. Создание такой программы на PHP позволит повторно использовать код, уже разработанный для основного приложения – форума. Это не только экономит время, но и помогает избежать излишних расходов на поддержку в будущем.

Расширение PHP-GTK позволяет программам на PHP, написанным для командной строки, стать полноценными приложениями с графическим интерфейсом, или GUI-приложениями. У них может быть, кроме того, общий код с веб-приложениями на PHP и программами, написанными для командной строки. Как и PHP, PHP-GTK не зависит от платформы, поэтому один и тот же код будет работать и в UNIX, и в Windows.

PHP, собранный как приложение, для выполнения в режиме CGI-программы может быть запущен и из командной строки. Чтобы выполнить определенный сценарий, его имя надо передать в качестве аргумента:

```
% php scan-discussions.php
```

В операционной системе UNIX можно также посредством синтаксиса «решетка-восклицание» в начале сценария явно указать запуск интерпретатора PHP для обработки. Если интерпретатор PHP находится в */usr/local/bin*, то первая строка сценария должна быть такой:

```
#!/usr/local/bin/php
```

Теперь можно запустить сценарий, просто введя его имя в командной строке, если файл сценария обладает правом на выполнение.

При запуске сценариев PHP из командной строки почти всегда указывают флаг `-q`, запрещающий PHP печатать HTTP-заголовки ответа в начале вывода:

```
% php -q scan-discussions.php
```

Это же можно сделать и так:

```
#!/usr/local/bin/php -q
```

Еще один полезный параметр командной строки – флаг `-c`, позволяющий указать альтернативный файл *php.ini* для загрузки конфигурации. Если по умолчанию файл инициализации PHP – */usr/local/lib/php.ini*, то может быть полезным иметь отдельный конфигурационный файл */usr/local/lib/php-commandline.ini* с такими установками, как `max_execution_time = 0`; это гарантирует, что ваш сценарий не вылетит через 30 секунд. Ниже показано, как использовать альтернативный файл:

```
% php -q -c /usr/local/lib/php-commandline.ini scan-discussions.php
```

Альтернативный вариант в начале сценария выглядит так:

```
#!/usr/local/bin/php -q -c /usr/local/lib/php-commandline.ini
```

Если вы собираетесь использовать часть ваших классов и функций и для веб, и для командной строки, то выделите код, который должен вести себя по-разному в этих различных условиях, например вывод в HTML-виде простого текста или доступ к переменным окружения, установленных веб-сервером. Полезно также снабдить программу глобальной переменной с именем `$COMMAND_LINE`. Присвойте ей значение `true` в начале вашего сценария командной строки. Тогда можно будет разветвить поведение сценария следующим образом:

```
if ($GLOBALS['COMMAND_LINE']) {
  print "Database error: ".mysql_error()."\n";
} else {
  print "Database error.<br>";
  error_log(mysql_error());
}
```

Этот код настраивает не только форматирование вывода в зависимости от контекста, в котором он выполняется (`\n` вместо `<br>`), но и направление потока информации. Тому, кто запускает программу из командной строки, полезно увидеть сообщение об ошибке от MySQL, но если это веб-приложение, то вряд ли желательно показывать пользователям потенциально секретные данные. Вместо этого программа выводит общее сообщение об ошибке, а подробности записывает в серверный журнал ошибок для конфиденциального просмотра.

Начиная с версии 4.3 реализация PHP включает в себя предварительно скомпилированную версию интерпретатора для командной строки (CLI).[1] Интерфейс CLI подобен варианту сборки PHP как CGI-приложения, но содержит некоторые существенные отличия, которые делают его более дружественным именно к работе в командной среде. Часть конфигурационных директив CLI имеют жестко зашитые в код значения. Например, директива `html_errors` установлена в `false`, а `implicit_flush` установлена в `true`. Директива `max_execution_time` установлена в 0, разрешая неограниченное время выполнения программы. Наконец, директива `register_argc_argv` установлена в `true`. Это означает, что информацию об аргументах можно увидеть в `$argv` и `$argc` вместо `$_SERVER['argv']` и `$_SERVER['argc']`. Обработка аргументов обсуждается в рецептах 20.1 и 20.2.

Интерпретатор CLI работает с несколько отличным от варианта CGI-сборки множеством аргументов. Так, он не поддерживает флаги `-q` или `-C`, поскольку по умолчанию выполняет именно те действия, которые задаются этими флагами. В то время как `-q` дает указание интерпретатору PHP (собранному как CGI) не выводить предопределенные HTTP-заголовки, интерпретатор CLI изначально никогда не печатает эти заголовки. Даже функция `header()` ничего не выводит в CLI. Точно так же флаг `-C` дает указание интерпретатору CGI не изменять каталог, в котором запускается сценарий. CLI-версия никогда не меняет каталог сценария.

Интерпретатор CLI также принимает и один новый аргумент: `-r`. Если за ним следует некоторый PHP-код без тегов сценария `<?php` и `?>`, то CLI запускает его как самостоятельную программу. Например, ниже показано, как напечатать текущее время:

```
% php -r 'print strftime("%c");'
```

Наконец, интерпретатор CLI определяет дескрипторы стандартных потоков ввода/вывода в виде констант `STDIN`, `STDOUT` и `STDERR`. Программист может работать с ними, вместо того чтобы создавать с помощью функции `fopen()` собственные файловые дескрипторы:

```
// читаем из стандартного потока ввода
$input = fgets(STDIN,1024);

// записываем в стандартный поток вывода
fwrite(STDOUT,$jokebook);

// записываем в стандартный поток ошибок
fwrite(STDERR,$error_code);
```

Те, кто работают с интерпретатором CLI, могут проверить, где запускается сценарий – в веб-контексте или в контексте командной строки с помощью функции `php_sapi_name()` вместо `$GLOBALS['COMMAND_LINE']`:

---

[1] Интерпретатор CLI можно построить и для версий 4.2.x, явным образом конфигурируя PHP с помощью параметра `--enable-cli`.

```
if ('cli' == php_sapi_name()) {
  print "Database error: ".mysql_error()."\n";
} else {
  print "Database error.<br>";
  error_log(mysql_error());
}
```

Программы, использующие расширение PHP-GTK, можно запускать как посредством интерпретатора CLI, так и сборки в режиме CGI PHP-GTK. Это расширение служит интерфейсом к пакету инструментальных средств разработки GTK+, представляющего собой библиотеку элементов оконного интерфейса, программ рисования экрана и других функций, необходимых для построения GUI-приложений.

Элементы оконного интерфейса – это элементы GUI-интерфейса, такие как кнопки, линейки прокрутки, окна, меню и диалоговые окна. Для построения приложения PHP-GTK программа должна уметь создавать такие элементы и располагать их на экране. В рецепте 20.5 показано, как создать и отобразить простое окно, в рецепте 20.6 – как расставить несколько элементов управления внутри окна для их совместного использования, а рецепт 20.8 объясняет, как работать с панелью меню.

Элементы управления окном взаимодействуют друг с другом и с остальной программой с помощью сигналов. Когда с элементом управления что-нибудь происходит, он выдает сигнал – например, когда по кнопке щелкают, она выдает сигнал `clicked`. В рецепте 20.7 обсуждается, как перехватывать эти сигналы и какие предпринимать действия в ответ на какой-нибудь сигнал. Простое приложение в рецепте 20.10 объединяет PHP-GTK с вызовами некоторых функций SOAP для показа погодных условий по всему миру.

Чтобы инсталлировать PHP-GTK в UNIX, загрузите последнюю версию PHP-GTK с *http://gtk.php.net/download.php* и библиотеку GTK+ с *http://www.gtk.org/download*. Понадобятся также *libtool* 1.4.2, *automake* 1.4 и *autoconf* 2.13 (доступные на *http://www.gnu.org/directory/*, если они еще не инсталлированы в вашей системе).

Загрузив все необходимые файлы и установив вспомогательные библиотеки и инструментальные средства разработки, распакуйте исходный дистрибутив PHP-GTK. В каталоге PHP-GTK запустите программу *./buildconf* для создания файлов конфигурации, программу *./configure* для создания make-файла, а затем программу *make* для сборки расширения PHP-GTK. Наконец, запустите *make install* для инсталляции расширения PHP-GTK в каталог расширений PHP. Подробные инструкции по инсталляции в UNIX, включая основные проблемы, можно найти на *http://gtk.php.net/manual/en/install.unix.php*.

Для инсталляции PHP-GTK в Windows в компиляции нет необходимости. С *http://gtk.php.net/download.php* можно загрузить предварительно скомпилированную версию расширения PHP-GTK и вспомогатель-

ные библиотеки. Загрузив и распаковав дистрибутив для Windows, скопируйте файлы, находящиеся в подкаталоге *php4,* в ваш каталог PHP, где размещены двоичные файлы (или создайте его, если он еще не существует). Скопируйте файлы из подкаталога *winnt\system32* в системный каталог *system32* (*C:\WINNT\SYSTEM32* для Windows NT и Windows 2000; *C:\WINDOWS\ SYSTEM32* для Windows 95 и Windows 98). Если файл *php.ini* еще не находится на своем месте, то скопируйте файл *winnt\php.ini* в каталог системы Windows (*C:\WINNT* или *C:\WINDOWS*). Если файл *php.ini* уже на месте, добавьте в него эти строки:

```
[PHP-GTK]
php-gtk.extensions = php_gtk_libglade.dll, php_gtk_sqpane.dll
```

Подробные инструкции по установке PHP в Windows находятся на *http://gtk.php.net/manual/en install.win32.php*.

В любой платформе после инсталляции расширения PHP-GTK необходимо вызывать функцию `dl()` для его загрузки в любой сценарий, в котором вы хотите использовать функциональность GTK. В Windows:

```
if (! class_exists('gtk')) {
    dl('php_gtk.dll');
}
```

В UNIX:

```
if (! class_exists('gtk')) {
    dl('php_gtk.so');
}
```

Для того чтобы один и тот же сценарий работал без изменений и в UNIX, и в Windows, можно загружать расширение PHP-GTK следующим образом:

```
if (! class_exists('gtk')) {
   dl('php_gtk.'. (((strtoupper(substr(PHP_OS,0,3))) == 'WIN')?'dll':'so'));
}
```

GTK+ представляет собой большой и мощный пакет инструментальных средств разработки. Он облегчает создание и работу с объектами GTK+, но разработка и планирование GUI-приложений – по-прежнему важная задача. В дополнение к обширной документации по PHP-GTK на *http://gtk.php.net/manual/* помощь можно найти и в документации собственно по GTK+ на *http://developer.gnome.org/doc/API/gtk/index.html*. Имена классов и функций C в документации по GTK+ почти точно соответствуют их эквивалентам в PHP. Кроме того, хотя учебное пособие на *http://www.gtk.org/tutorial/* написано для GTK+ 2.0 (не для 1.2), это все равно хорошее введение в основы и практику построения приложений GTK+.

20.1. Анализ аргументов программы

Задача

Необходимо обработать аргументы, переданные в командной строке.

Решение

Количество переданных программе аргументов можно посмотреть в переменной `$_SERVER['argc']`, а их значения — в переменной `$_SERVER['argv']`. Первый аргумент, `$_SERVER['argv'][0]`, — это имя выполняемого сценария:

```
if ($_SERVER['argc'] != 2) {
    die("Wrong number of arguments: I expect only 1.");
}

$size = filesize($_SERVER['argv'][1]);

print "I am $_SERVER[argv][0] and report that the size of ";
print "$_SERVER[argv][1] is $size bytes.";
```

Обсуждение

Для того чтобы установить параметры на основе флагов, переданных в командной строке, выполните цикл по массиву `$_SERVER['argv']` от 1 до `$_SERVER['argc']`:

```
for ($i = 1; $i < $_SERVER['argc']; $i++) {
    switch ($_SERVER['argv'][$i]) {
    case '-v':
        // устанавливаем флаг
        $verbose = 1;
        break;
    case '-c':
        // переходим к следующему аргументу
        $i++;
        // если он установлен, записываем значение
        if (isset($_SERVER['argv'][$i])) {
            $config_file = $_SERVER['argv'][$i];
        } else {
            // выходим, если не указано имя файла
            die("Must specify a filename after -c");
        }
        break;
    case '-q':
        $quiet = 1;
        break;
    default:
        die('Unknown argument: '.$_SERVER['argv'][$i]);
        break;
    }
}
```

В этом примере аргументы -v и -q представляют собой флаги, устанавливающие $verbose и $quiet, но за аргументом -c ожидается строка. Эта строка присваивается переменной $config_file.

См. также

Рецепт 20.2 о дополнительном анализе аргументов с помощью *getopt*; документацию по $_SERVER['argc'] и $_SERVER['argv'] на *http://www.php.net/reserved.variables*.

20.2. Анализ аргументов программы с помощью getopt

Задача

Необходимо проанализировать параметры программы, которые могут быть представлены как в сокращенном, так и в расширенном синтаксисе, или могут быть сгруппированы.

Решение

Это делается с помощью PEAR-класса Console_Getopt. Его метод getopt() может анализировать короткие параметры, такие как -a или -b, и длинные — --alice или --bob:

```
$o = new Console_Getopt;

// принимает -a, -b и -c
$opts = $o->getopt($_SERVER['argv'],'abc');

// принимает --alice и --bob
$opts = $o->getopt($_SERVER['argv'],'',array('alice','bob'));
```

Обсуждение

Для того чтобы проанализировать короткие параметры, передайте функции Console_Getopt::getopt() массив аргументов командной строки и строку с указанием допустимых параметров. Этот пример допускает -a, -b или -c в качестве аргументов, поодиночке или в группах:

```
$o = new Console_Getopt;
$opts = $o->getopt($_SERVER['argv'],'abc');
```

Для предыдущей строки параметров abc допустимыми для передачи множествами параметров являются:

```
% program.php -a -b -c
% program.php -abc
% program.php -ab -c
```

Метод getopt() возвращает массив. Первый элемент массива представляет собой список всех проанализированных параметров, указанных

в командной строке, вместе со своими значениями. Второй элемент – это любой указанный параметр командной строки, отсутствовавший в определении аргументов, переданных методу `getopt()`. Например, если предыдущая программа запускается как:

```
% program.php -a -b sneeze
```

то `$opts` имеет вид:

```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => a
                    [1] =>
                )
            [1] => Array
                (
                    [0] => b
                    [1] =>
                )
        )
    [1] => Array
        (
            [0] => program.php
            [1] => sneeze
        )
)
```

Поставьте двоеточие после параметра в строке определения, указывая тем самым, что параметр должен иметь значение. Два двоеточия означают, что значение может отсутствовать. Поэтому строка `ab:c::` означает, что `a` не может иметь значения, `b` должна иметь значение, а `c` может принимать значение, если оно указано. При такой строке определения программа, запущенная как:

```
% program.php -a -b sneeze
```

выдает `$opts`:

```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => a
                    [1] =>
                )
            [1] => Array
                (
```

```
                    [0] => b
                    [1] => sneeze
                )
            )
        [1] => Array
            (
                [0] => program.php
            )
    )
```

Поскольку sneeze теперь считается значением параметра b, ее больше нет в массиве непроанализированных параметров. Обратите внимание, что массив непроанализированных параметров всегда содержит имя программы.

Для того чтобы проанализировать длинные аргументы, надо передать методу getopt() массив, описывающий требуемые аргументы. Поместите каждый аргумент в элемент массива (не включайте начальные символы --) и завершите его символом = для обозначения обязательного аргумента или символом == для обозначения необязательного аргумента. Этот массив представляет третий аргумент метода getopt(). Второй аргумент (строка для коротких аргументов) может быть оставлен пустым или нет, в зависимости от необходимости анализа коротких аргументов. Следующий пример позволяет использовать debug в качестве аргумента без значения, name с обязательным значением, а size с необязательным значением:

```
require 'Console/Getopt.php';
$o = new Console_Getopt;
$opts = $o->getopt($_SERVER['argv'],'',array('debug','name=','size=='));
```

Вот допустимые способы запуска этой программы:

```
% program.php --debug
% program.php --name=Susannah
% program.php --name Susannah
% program.php --debug --size
% program.php --size=56 --name=Susannah
% program.php --name --debug
```

Последний вариант вызова допустим (хотя и не производителен), поскольку он рассматривает --debug как значение аргумента name и не считает, что аргумент debug должен быть установлен. Значения могут быть отделены от своих аргументов символом = или пробелом.

Для длинных аргументов метод getopt() включает начальные символы -- в массив проанализированных аргументов; например, при таком способе запуска:

```
% program.php --debug --name=Susannah
```

$opts устанавливается в:

```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => --debug
                    [1] =>
                )
            [1] => Array
                (
                    [0] => --name
                    [1] => Susannah
                )
        )
    [1] => Array
        (
            [0] => program.php
        )
)
```

Мы использовали в качестве массива аргументов командной строки `$_SERVER['argv']`, который хорош по умолчанию. Класс `Console_Getopt` предоставляет и другой метод, `readPHPArgv()`, для поиска аргументов командной строки в `$argv` и `$HTTP_SERVER_VARS['argv']`. Его результаты надо передать методу `getopt()`:

```
require 'Console/Getopt.php';
$o = new Console_Getopt;
$opts = $o->getopt($o->readPHPArgv(),'',array('debug','name=','size==='));
```

И метод `getopt()` и метод `readPHPArgv()`, столкнувшись с ошибкой, возвращают объект `Getopt_Error`, например, когда нет значения для аргумента, которому оно необходимо. Класс `Getopt_Error` расширяет базовый класс `PEAR_Error`, поэтому для обработки ошибок можно применять хорошо знакомые методы:

```
require 'Console/Getopt.php';
$o = new Console_Getopt;
$opts = $o->getopt($o->readPHPArgv(),'',array('debug','name=','size==='));

if (PEAR::isError($opts)) {
    print $opts->getMessage();
} else {
    // обрабатываем опции
}
```

См. также

Рецепт 20.1 об анализе аргументов программы без *getopt*; документацию по `Console_Getopt` на *http://pear.php.net/manual/en/core.console.getopt.php*.

20.3. Чтение ввода с клавиатуры

Задача

Необходимо прочитать клавиатурный ввод пользователя.

Решение

Для этого нужна функция `fopen()` со специальным именем файла *php://stdin*:

```
print "Type your message. Type '.' on a line by itself when you're done.\n";

$fh = fopen('php://stdin','r') or die($php_errormsg);
$last_line = false;  $message = '';
while (! $last_line) {
    $next_line = fgets($fp,1024);
    if (".\n" == $next_line) {
      $last_line = true;
    } else {
      $message .= $next_line;
    }
}

print "\nYour message is:\n$message\n";
```

Если инсталлировано расширение Readline, вызывайте функцию `readline()`:

```
$last_line = false; $message = '';
while (! $last_line) {
    $next_line = readline();
    if ('.' == $next_line) {
        $last_line = true;
    } else {
        $message .= $next_line."\n";
    }
}

print "\nYour message is:\n$message\n";
```

Обсуждение

Получив с помощью функции `fopen()` дескриптор файла, указывающего на *stdin*, для обработки ввода можно использовать все стандартные функции чтения файла (`fread()`, `fgets()` и т. д.). В данном решении задействована функция `fgets()`, за один раз возвращающая одну строку ввода. Если вызывается функция `fread()`, то ввод все равно должен завершаться символом новой строки, чтобы заставить функцию `fread()` вернуть значение. Например, если выполнить код:

```
$fh = fopen('php://stdin','r') or die($php_errormsg);
$msg = fread($fh,4);
print "[$msg]";
```

и ввести строку `tomato`, а затем символ новой строки, то на выходе получится строка `[toma]`. Функция `fread()`, как указано, забирает из *stdin* только четыре символа, но ей нужен символ новой строки – как сигнал выхода из ожидания ввода с клавиатуры.

Расширение Readline предоставляет интерфейс к библиотеке GNU Readline. Функция `readline()` возвращает строку за один раз без символа новой строки в конце. Readline позволяет редактировать строки в стиле Emacs и *vi*. Ее можно применять для хранения истории введенных ранее команд:

```
$command_count = 1;
while (true) {
    $line = readline("[$command_count]--> ");
    readline_add_history($line);
    if (is_readable($line)) {
        print "$line is a readable file.\n";
    }
    $command_count++;
}
```

Этот пример показывает приглашение с увеличивающимся счетчиком перед каждой строкой. Поскольку каждая строка добавляется к истории прочитанных строк с помощью функции `readline_add_history()`, нажатие стрелок вверх и вниз позволяет прокручивать введенные строки.

См. также

Документацию по функции `fopen()` на *http://www.php.net/fopen*, по функции `fgets()` на *http://www.php.net/fgets*, по функции `fread()` на *http://www.php.net/fread* и по расширению Readline на *http://www.php.net/readline*; информацию о библиотеке Readline на *http://cnswww.cns.cwru.edu/php/chet/ readline/rltop.html*.

20.4. Чтение паролей

Задача

Необходимо прочитать информацию из командной строки, не отображая при этом сам ввод (например, при вводе паролей).

Решение

В системах UNIX отображение вводимых символов достигается применением */bin/stty*:

```
// выключаем эхо
`/bin/stty -echo`;

// читаем пароль
```

```
$password = readline();

// снова включаем эхо
`/bin/stty echo`;
```

В Windows применяется функция `w32api_register_function()` — для импортирования функции `_getch()` из *msvcrt.dll*:

```
// загружаем расширение w32api и регистрируем функцию _getch()
dl('php_w32api.dll');
w32api_register_function('msvcrt.dll','_getch','int');

while(true) {
    // получаем символ с клавиатуры
    $c = chr(_getch());
    if ( "\r" == $c ||  "\n" == $c ) {
        // если это символ новой строки, прерываем цикл,
        // мы получили наш пароль
        break;
    } elseif ("\x08" == $c) {
        /* если это символ забоя, удалите предыдущий символ
           из переменной $password */
        $password = substr_replace($password,'',-1,1);
    } elseif ("\x03" == $c) {
       // если это Control-C, очищаем переменную $password и выходим из цикла
        $password = NULL;
        break;
    } else {
        // в противном случае добавляем символ к паролю
        $password .= $c;
    }
}
```

Обсуждение

В UNIX для управления характеристиками терминала применяется */bin/stty*, предотвращающая отображение введенных символов на экране во время чтения пароля. В Windows нет */bin/stty*, поэтому для получения доступа к функции `_getch()` из динамической C-библиотеки Microsoft *msvcrt.dll* применяется расширение W32api. Функция `_getch()` читает символ, не отображая его на экране. Она возвращает ASCII-код прочитанного символа, поэтому его надо конвертировать в символьную строку с помощью функции `chr()`. Затем предпринимаются действия в зависимости от типа напечатанного символа. Если это символ новой строки или возврата каретки, то выходите из цикла, т. к. пароль уже введен. Если это символ забоя (backspace), то удаляете последний символ пароля. Если это символ прерывания `<Control+C>`, то вы устанавливаете пароль в `NULL` и выходите из цикла. Если ни одно из этих условий не выполняется, то символ добавляется к переменной `$password`. После выхода из цикла переменная `$password` содержит введенный пароль.

Следующая программа показывает приглашения `Login:` и `Password:` и сравнивает введенный пароль с соответствующим зашифрованным паролем, хранящимся в */etc/passwd*. При этом необходимо, чтобы в системе не использовались теневые пароли.

```
print "Login: ";
$fh = fopen('php://stdin','r')   or die($php_errormsg);
$username = rtrim(fgets($fh,64)) or die($php_errormsg);

preg_match('/^[a-zA-Z0-9]+$/',$username)
    or die("Invalid username: only letters and numbers allowed");


print 'Password: ';
`/bin/stty -echo`;
$password = rtrim(fgets($fh,64)) or die($php_errormsg);
`/bin/stty echo`;
print "\n";

// больше нечего читать с клавиатуры
fclose($fh);

// находим соответствующую строку в /etc/passwd
$fh = fopen('/etc/passwd','r')   or die($php_errormsg);
$found_user = 0;
while (! ($found_user || feof($fh))) {
    $passwd_line = fgets($fh,256);
    if (preg_match("/^$username:/",$passwd_line)) {
        $found_user = 1;
    }
}
fclose($fh);

$found_user or die ("Can't find user \"$username\"");

// анализируем корректную строку из /etc/passwd
$passwd_parts = split(':',$passwd_line);

/* кодируем введенный пароль и сравниваем его с паролем в
   /etc/passwd */
$encrypted_password = crypt($password,
                            substr($passwd_parts[1],0,CRYPT_SALT_LENGTH));

if ($encrypted_password == $passwd_parts[1]) {
    print "login successful";
} else {
    print "login unsuccessful";
}
```

См. также

Документацию по функции `readline()` на *http://www.php.net/readline*, по функции `chr()` на *http://www.php.net/chr*, по функции `w32api_register_function()` на *http://www.php.net/w32api-register-function* и по функции `_getch()` на *http://msdn.microsoft.com/library/en-us/vccore98/*

HTML/_crt__getch.2c_._getche.asp; в UNIX см. страницу помощи *stty(1)* программы man вашей системы.

20.5. Показ в окне графических элементов управления

Задача

Необходимо показать окно с GUI-элементами управления в нем, например с кнопкой.

Решение

Создайте окно, создайте элемент управления, а затем добавьте элемент управления в окно:

```
// создаем окно
$window = &new GtkWindow();

// создаем кнопку и добавляем ее к окну
$button = &new GTKButton('Click Me, Alice');
$window->add($button);

// показываем окно
$window->show_all();

// необходимо обеспечить корректное завершение программы
function shutdown() { gtk::main_quit(); }
$window->connect('destroy','shutdown');

// запускаем цикл, обрабатывающий сигнал GTK
gtk::main();
```

Обсуждение

Сначала создайте окно путем создания нового экземпляра класса `GtkWindow`. Объект GTK должен создаваться по ссылке: `&new GtkWindow()`, а не `new GtkWindow()`. Затем создайте новый объект `GtkButton` с меткой «Click Me, Alice». Передавая переменную `$button` методу окна `add()`, добавьте кнопку к окну. Метод `show_all()` показывает окно и все элементы управления в нем. В этом примере единственным элементом управления в окне является кнопка. Следующие две строки обеспечивают выход из программы при закрытии окна. Функция `shutdown()` представляет собой функцию обратной связи приложения, которая объясняется позже, в рецепте 20.7.

Последняя строка необходима во всех программах PHP-GTK. Вызов функции `gtk::main()` начинает цикл обработки сигнала. Это означает, что программа ожидает сигналы, сгенерированные элементами управления GUI, а затем отвечает на эти сигналы по мере их поступления. Эти сигналы представляют такую активность, как нажатие на кнопки, изменение размеров окон и ввод в текстовых окнах. Единственный

сигнал, на который обращает внимание именно эта программа, это сигнал `destroy`. Когда пользователь закрывает главное окно программы, выдается сигнал `destroy` и вызывается функция `gtk::main_quit()`. Эта функция завершает программу.

См. также

Документацию по классу `GtkWindow` на *http://gtk.php.net/manual/en/gtk.gtkwindow.php*, по функции `GTKContainer::add()` на *http://gtk.php.net/manual/en/gtk.gtkcontainer.method.add.php*, по функции `GtkWidget::show_all()` на *http://gtk.php.net/manual/en/gtk.gtkwidget.method.show_all.php*, по классу `GtkButton` на *http://gtk.php.net/manual/en/gtk.gtkbutton.php*, по функции `gtk::main_quit()` на *http://gtk.php.net/manual/en/gtk.method.main_quit.php* и по функции `gtk::main()` на *http://gtk.php.net/manual/en/gtk.method.main.php*; учебное пособие на *http://gtk.php.net/manual/en/tutorials.hellow.php* – полезное введение в основы GTK-программирования GTK.

20.6. Показ в окне нескольких графических элементов управления

Задача

Необходимо показать в окне более одного элемента управления.

Решение

Добавьте все элементы управления в контейнер, а затем добавьте контейнер к окну:

```
// создаем окно
$window = &new GtkWindow();

// создаем контейнер - GtkVBox, который выравнивает элементы
// управления по вертикали
$container = &new GtkVBox();

// создаем текстовый элемент управления и добавляем его в контейнер
$text_entry = &new GtkEntry();
$container->pack_start($text_entry);

// создаем кнопку и добавляем ее в контейнер
$a_button = &new GtkButton('Abort');
$container->pack_start($a_button);

// создаем другую кнопку и добавляем ее в контейнер
$r_button = &new GtkButton('Retry');
$container->pack_start($r_button);

// создаем еще одну кнопку и добавляем ее в контейнер
$f_button = &new GtkButton('Fail');
$container->pack_start($f_button);
```

```
// добавляем контейнер к окну
$window->add($container);

// показываем окно
$window->show_all();

// необходимо обеспечить корректное завершение программы
function shutdown() { gtk::main_quit(); }
$window->connect('destroy','shutdown');

// запускаем цикл, обрабатывающий сигнал GTK
gtk::main();
```

Обсуждение

Окно представляет собой контейнер, который может содержать только один элемент управления. Чтобы разместить в окне несколько элементов управления, необходимо поместить все элементы управления в другой контейнер, который может содержать более одного элемента управления, а затем поместить такой контейнер в окно. Этот процесс может быть вложенным: элементы управления в контейнере и сами могут быть контейнерами.

В приведенном выше решении контейнеры добавляются в контейнер `GtkVBox`, выравнивающий дочерние элементы управления окном по вертикали, как показано на рис. 20.1. Добавлять элементы управления в `GtkVBox` можно и при помощи метода `add()`, но мы вместо него вызовем метод `pack_start()`, для того чтобы размер контейнера автоматически обновлялся при добавлении каждого нового элемента.

Рис. 20.1. *Элементы управления окном в GtkVBox*

`GtkHBox` подобен `GtkVBox`. Он выравнивает свои дочерние элементы управления по горизонтали, а не по вертикали. На рис. 20.2 показаны четыре элемента управления из раздела «Решение» в `CtkHBox`.

Рис. 20.2. *Элементы управления в GtkHBox*

Класс GtkTable – это контейнер с более гибкой схемой размещения; он выравнивает свои дочерние элементы по сетке:

```
// создаем окно
$window = &new GtkWindow();

// создаем контейнер с тремя строками и двумя столбцами
$container = &new GtkTable(3,2);

// создаем текстовый элемент управления и добавляем его в контейнер
$text_entry = &new GtkEntry();
$container->attach($text_entry,0,2,0,1);

// создаем кнопку и добавляем ее в контейнер
$a_button = &new GtkButton('Abort');
$container->attach($a_button,0,1,1,2);

// создаем другую кнопку и добавляем ее в контейнер
$r_button = &new GtkButton('Retry');
$container->attach($r_button,1,2,1,2);

// создаем еще одну кнопку и добавляем ее в контейнер
$f_button = &new GtkButton('Fail');
$container->attach($f_button,0,2,2,3);

// добавляем контейнер к окну
$window->add($container);

// показываем окно
$window->show_all();

// необходимо обеспечить корректное завершение программы
function shutdown() { gtk::main_quit(); }
$window->connect('destroy','shutdown');

// запускаем цикл, обрабатывающий сигнал GTK
gtk::main();
```

Элементы управления добавляются в контейнер GtkTable с помощью метода attach(). Первый аргумент метода attach() – это добавляемый элемент управления, а следующие четыре описывают расположение элемента в сетке. Второй и третий аргументы – это начальный и конечный столбцы для элемента управления, а четвертый и пятый – это начальная и конечная строки. Например:

```
$container->attach($text_entry,0,2,0,1)
```

означает, что элемент управления для ввода текста начинается в нулевом столбце, а заканчивается во втором, занимая два столбца. При этом он начинается в нулевой строке и заканчивается в первой, занимая только одну строку. Нумерация строк и столбцов начинается с нуля. Текстовый элемент управления и кнопка, выровненные в контейнере GtkTable, показаны на рис. 20.3.

Рис. 20.3. *Элементы управления в GtkTable*

См. также

Документацию по контейнерам на *http://gtk.php.net/manual/en/gtk.containers.whatare.php*, по классу `GtkVBox` на *http://gtk.php.net/manual/en/gtk.gtkvbox.php*, по классу `GtkHBox` на *http://gtk.php.net/manual/en/gtk.gtkhbox.php*, по функции `GtkBox::pack_start()` на *http://gtk.php.net/manual/en/gtk.gtkbox.method.pack_start.php*, по классу `GtkTable` на *http://gtk.php.net/manual/en/gtk.gtktable.php* и по функции `GtkTable::attach()` на *http://gtk.php.net/manual/en/gtk.gtktable.method.attach.php*.

20.7. Реакция на действия пользователя

Задача

Необходимо производить ответные действия, когда пользователь щелкает по кнопке, выбирает элемент из выпадающего списка или другим образом взаимодействует с GUI-элементами управления.

Решение

Напишите функцию обратного вызова, а затем свяжите ее с сигналом с помощью метода `connect()`:

```
// создаем окно
$window = &new GtkWindow();

// создаем кнопку с текущим временем в качестве ее метки
$button = &new GtkButton(strftime('%c'));

// установите функцию update_time() в качестве функции
// обратного вызова для сигнала «clicked»
$button->connect('clicked','update_time');

function update_time($b) {
    // текст кнопки находится в дочернем элементе кнопки -
    // элементе управления метка
    $b_label = $b->child;
    // set the label text to the current time
    $b_label->set_text(strftime('%c'));
}
```

```
// добавляем кнопку к окну
$window->add($button);

// показываем окно
$window->show_all();

// необходимо обеспечить корректное завершение программы
function shutdown() { gtk::main_quit(); }
$window->connect('destroy','shutdown');

// запускаем цикл, обрабатывающий сигнал GTK
gtk::main();
```

Обсуждение

Код в решении показывает окно с кнопкой в нем. На кнопке находится время, визуализированное с помощью `strftime('%c')`. Когда по кнопке щелкают, ее метка обновляется и отражает текущее время.

Функция `update_time()` вызывается всякий раз, когда щелкают по кнопке, поскольку метод `$button->connect('clicked','update_time')` связывает функцию обратного вызова `update_time()` с сигналом кнопки `clicked`. Первый аргумент функции обратного вызова представляет собой элемент управления, сигнал которого запускает вызов функции. В данном случае это означает, что переменная `$button` передается функции `update_time()`. Вы говорите функции `connect()` передать дополнительные параметры функции обратного вызова, передавая их функции `connect()` вслед за именем функции обратного вызова. Следующий пример показывает окно с кнопкой и отдельной меткой. Время выводится в метке и обновляется, когда щелкают по кнопке:

```
// создаем окно
$window = &new GtkWindow();

// создаем контейнер для метки и кнопки
$container = &new GtkVBox();

// создаем метку, показывающую время
$label = &new GtkLabel(strftime('%c'));

// добавляем метку в контейнер
$container->pack_start($label);

// создаем кнопку
$button = &new GtkButton('Update Time');

/* устанавливаем функцию update_time() в качестве функции обратного вызова
   для сигнала «clicked» и передаем ей переменную $label */
$button->connect('clicked','update_time',$label);

function update_time($b,$lb) {
    $lb->set_text(strftime('%c'));
}

// add the button to the container
```

```
$container->pack_start($button);

// добавляем контейнер к окну
$window->add($container);

// показываем окно
$window->show_all();

// необходимо обеспечить корректное завершение программы
function shutdown() { gtk::main_quit(); }
$window->connect('destroy','shutdown');

// запускаем цикл, обрабатывающий сигнал GTK
gtk::main();
```

Поскольку переменная `$label` находится в списке аргументов, переданных методу `$button->connect()`, то она также передается функции `update_time()`. Вызов функции `set_text()` для переменной `$label` обновляет текст, показываемый в метке.

См. также

Документацию по сигналам и функциям обратного вызова на *http://gtk.php.net/manual/en/gtk.signals.php*, по методу `GtkObject::connect()` на *http://gtk.php.net/manual/en/gtk.gtkobject.method.connect.php* и по сигналу `clicked` класса `GtkButton` на *http://gtk.php.net/manual/en/gtk.gtkbutton.signal.clicked.php*.

20.8. Показ меню

Задача

Необходимо показать панель меню в верхней части окна GTK-приложения.

Решение

Создаем `GtkMenu`. Создаем отдельный объект `GtkMenuItem` для каждого элемента меню, который вы хотите показать, и добавляем каждый элемент меню в `GtkMenu` с помощью функции `append()`. Затем создаем главное меню `GtkMenuItem` с меткой, которая должна появиться в панели меню (т. е. «File» или «Options»). Добавляем меню к главному меню с помощью функции `set_submenu()`. Создаем `GtkMenuBar` и добавляем главное меню в панель меню с помощью функции `append()`. Наконец, добавляем панель меню к окну:

```
// создаем окно
$window = &new GtkWindow();

// создаем меню
$menu = &new GtkMenu();

// создаем элемент меню и добавляем его в меню
```

```
$menu_item_1 = &new GtkMenuItem('Open');
$menu->append($menu_item_1);

// создаем следующий элемент меню и добавляем его в меню
$menu_item_2 = &new GtkMenuItem('Close');
$menu->append($menu_item_2);

// создаем еще один элемент меню и добавляем его в меню
$menu_item_2 = &new GtkMenuItem('Save');
$menu->append($menu_item_2);

// создаем главное меню и добавляем в него существующее меню
$root_menu = &new GtkMenuItem('File');
$root_menu->set_submenu($menu);

// создаем панель меню и добавляем в него главное меню
$menu_bar = &new GtkMenuBar();
$menu_bar->append($root_menu);

// добавляем панель меню к окну
$window->add($menu_bar);

// показываем окно
$window->show_all();

// необходимо обеспечить корректное завершение программы
function shutdown() { gtk::main_quit(); }
$window->connect('destroy','shutdown');

// запускаем цикл, обрабатывающий сигнал GTK
gtk::main();
```

Обсуждение

Меню вовлекает в иерархию совсем немного объектов. `GtkWindow` (или другой контейнер) содержит `GtkMenuBar`. `GtkMenuBar` содержит `GtkMenuItem` для каждого меню верхнего уровня в панели меню (т. е. «File», «Options» или «Help»). Каждый элемент меню верхнего уровня `GtkMenuItem` имеет `GtkMenu` в качестве подменю. Это подменю содержит свои элементы `GtkMenuItem`, которые выводятся ниже меню верхнего уровня.

Как и любой GTK-элемент управления, объект `GtkMenuItem` может иметь функции обратного вызова, обрабатывающие сигналы. Когда элемент меню выбран, он выдает сигнал `activate`. Чтобы предпринять какие-либо действия в ответ на выбор элемента меню, свяжите его сигнал `activate` с функцией обратного вызова. Ниже приведена версия программы показа времени с кнопкой и меткой из рецепта 20.7 с двумя элементами меню – «Update», обновляющим время в метке, и «Quit», завершающим программу:

```
// создаем окно
$window = &new GtkWindow();

// создаем контейнер для метки и кнопки
$container = &new GtkVBox();
```

```
// создаем меню
$menu = &new GtkMenu();

// создаем элемент меню и добавляем его в меню
$menu_item_1 = &new GtkMenuItem('Update');
$menu->append($menu_item_1);

// создаем другой элемент меню и добавляем его в меню
$menu_item_2 = &new GtkMenuItem('Quit');
$menu->append($menu_item_2);

// создаем главное меню и добавляем в него существующее меню
$root_menu = &new GtkMenuItem('File');
$root_menu->set_submenu($menu);

// создаем панель меню и добавляем в него главное меню
$menu_bar = &new GtkMenuBar();
$menu_bar->append($root_menu);

// добавляем меню в контейнер
$container->add($menu_bar);

// создаем метку, показывающую время
$label = &new GtkLabel(strftime('%c'));

// добавляем метку в контейнер
$container->pack_start($label);

// создаем кнопку
$button = &new GtkButton('Update Time');

/* устанавливаем функцию update_time() в качестве функции обратного вызова
   для сигнала «clicked» и передаем ей переменную $label */
$button->connect('clicked','update_time',$label);

function update_time($b,$lb) {
    $lb->set_text(strftime('%c'));
}

// добавляем кнопку в контейнер
$container->pack_start($button);

// если выбран элемент меню Update, то вызываем функцию update_time()
$menu_item_1->connect('activate','update_time',$label);

// если выбран элемент меню Quit, то выходим
$menu_item_2->connect('activate','shutdown');

// добавляем контейнер к окну
$window->add($container);

// показываем окно
$window->show_all();

// необходимо обеспечить корректное завершение программы
function shutdown() { gtk::main_quit(); }
$window->connect('destroy','shutdown');
```

```
// запускаем цикл, обрабатывающий сигнал GTK
gtk::main();
```

Функции обратного вызова связываются с элементами меню с помощью их методов `connect()`. Связь же между функциями обратного вызова и сигналами `activate` устанавливается ближе к концу программы, поскольку именно вызов `$menu_item_1->connect()` передает переменную `$label` функции `update_time()`. Для успешной передачи переменной `$label` функции `update_time()` в процессе выполнения программы необходимо вызывать функцию `connect()` уже после того, как переменная `$label` получит свое значение.

См. также

Документацию по классу `GtkMenu` на *http://gtk.php.net/manual/en/gtk.gtkmenu.php*, по функции `GtkMenuShell::append()` на *http://gtk.php.net/manual/en/gtk.gtkmenushell.method.append.php*, по классу `GtkMenuItem` на *http://gtk.php.net/manual/en/gtk.gtkmenuitem.php*, по функции `GtkMenuItem::set_submenu()` на *http://gtk.php.net/manual/en/gtk.gtkmenuitem.method.set_submenu.php*, по сигналу `activate` класса `GtkMenuItem` на *http://gtk.php.net/manual/en/gtk.gtkmenuitem.signal.activate.php* и по классу `GtkMenuBar` на *http://gtk.php.net/manual/en/gtk.gtkmenubar.php*.

20.9. Программа: Командная оболочка

Программа *command-shell.php*, показанная в примере 20.1, выдает приглашение, напоминающее приглашение командной оболочки, которое позволяет интерактивно выполнить PHP-программу. Она читает построчно с помощью функции `readline()`, а затем выполняет программу с помощью функции `eval()`. По умолчанию она выполняет каждую строку сразу после ее ввода. Однако в многострочном режиме (указываемом с помощью параметров `-m` или `--multiline`) она сохраняет прочитанные строки, пока вы не введете символ . в отдельной строке; затем она запускает накопленный программный код.

Кроме того, программа *command-shell.php* использует возможности Readline в части расширения слов, чтобы облегчить ввод PHP-функций. Введите несколько символов и нажмите клавишу Tab, чтобы посмотреть список функций, соответствующих введенным вами символам.

Эта программа полезна для интерактивного выполнения фрагментов кода или тестирования различных команд. Переменные, функции и классы, определенные в каждой строке кода, остаются определенными до выхода из программы, поэтому можно тестировать различные запросы к базе данных, например:

```
% php -q command-shell.php
[1]> require 'DB.php';

[2]> $dbh = DB::connect('mysql://user:pwd@localhost/phpc');
```

```
[3]> print_r($dbh->getAssoc('SELECT sign,planet,start_day
             FROM zodiac WHERE element LIKE "water"'));
Array
(
    [Cancer] => Array
        (
            [0] => Moon
            [1] => 22
        )
    [Scorpio] => Array
        (
            [0] => Mars
            [1] => 24
        )
    [Pisces] => Array
        (
            [0] => Neptune
            [1] => 19
        )
)
```

Код программы *command-shell.php* показан в примере 20.1.

Пример 20.1. command-shell.php

```
// Загружаем библиотеку readline
if (! function_exists('readline')) {
    dl('readline.'. (((strtoupper(substr(PHP_OS,0,3))) == 'WIN')?'dll':'so'))
        or die("Readline library required\n");
}

// Загружаем класс Console_Getopt
require 'Console/Getopt.php';

$o = new Console_Getopt;
$opts = $o->getopt($o->readPHPArgv(),'hm',array('help','multiline'));

// Выходим с традиционным сообщением, если аргументы неправильные
if (PEAR::isError($opts)) {
    print $opts->getMessage();
    print "\n";
    usage();
}

// поведение по умолчанию - это оценка каждой команды
// в том виде, как она введена
$multiline = false;

foreach ($opts[0] as $opt) {
    // удаляем все начальные -s
    $opt[0] = preg_replace('/^-+/','',$opt[0]);

    // проверяем первый символ аргумента
    switch($opt[0][0]) {
```

```
    case 'h':
        // показываем подсказку
        usage();
        break;
    case 'm':
        $multiline = true;
        break;
    }
}

// устанавливаем показ ошибки
ini_set('display_errors',false);
ini_set('log_errors',true);

// строим таблицу расширения readline
$functions = get_defined_functions();
foreach ($functions['internal'] as $k => $v) {
    $functions['internal'][$k] = "$v(";
}
function function_list($line) {
    return $GLOBALS['functions']['internal'];
}
readline_completion_function('function_list');

$cmd = '';
$cmd_count = 1;

while (true) {
    // получаем строку ввода пользователя
    $s = readline("[$cmd_count]> ");
    // добавляем ее в историю команд
    readline_add_history($s);
    // если мы в многострочном режиме:
    if ($multiline) {
        // если было введено только «.»
        if ('.' == rtrim($s)) {
            // выполняем код с помощью функции eval()
            eval($cmd);
            // удаляем накопленный программный код
            $cmd = '';
            // увеличиваем счетчик команд
            $cmd_count++;
            // выдаем следующее приглашение в новой строке
            print "\n";
        } else {
           /* в противном случае добавляем новую строку (присоединяем символ
              новой строки) к накопленному коду,благодаря чему остальные
              введенные строки не превращаются в комментарии типа //
           */
            $cmd .= $s."\n";;
        }
    } else {
        // если мы не в многострочном режиме,
```

```
        // то выполняем строку с помощью функции eval()
        eval($s);
        // увеличиваем счетчик команд
        $cmd_count++;
        // выдаем следующее приглашение в новой строке
        print "\n";
    }
}

// показываем стандартную информацию подсказки
function usage() {
    $my_name = $_SERVER['argv'][0];

    print<<<_USAGE_
Usage: $my_name [-h|--help] [-m|--multiline]

  -h, --help: display this help
  -m, --multiline: execute accumulated code when "." is entered
                   by itself on a line. The default is to execute
                   each line after it is entered.

_USAGE_;
    exit(-1);
}
```

20.10. Программа: Служба погоды

Программа *gtk-weather.php*, показанная в примере 20.2, использует механизм SOAP и погодную веб-службу для информирования о погоде по всему миру. В своем интерфейсе она объединяет многочисленные элементы управления: меню, ускоряющие клавиши, кнопки, окна текстового ввода, метки, окна с прокруткой и колоночные списки.

Чтобы использовать программу *gtk-weather.php*, сначала найдите станции погоды, введя условия поиска в текстовом окне, и щелкните по кнопке Search. Поиск станций погоды показан на рис. 20.4.

После получения списка станций погоды можно увидеть погодные условия на определенной станции, выбрав станцию и щелкнув по кнопке Add. Код станции и ее текущие погодные условия добавляются к списку внизу окна. Можно снова искать и добавлять новые станции в список. Окно программы *gtk-weather.php* с несколькими добавленными станциями показано на рис. 20.5.

Веб-служба, которую использует эта программа, называется *GlobalWeather*; дополнительную информацию о ней см. на *http://www.capescience.com/webservices/globalweather/index.shtml*.

Пример 20.2. gtk-weather.php

```
// Загружаем расширение GTK
dl('php_gtk.'. (((strtoupper(substr(PHP_OS,0,3))) == 'WIN')?'dll':'so'));

// Загружаем класс SOAP-клиента
require 'SOAP/Client.php';
```

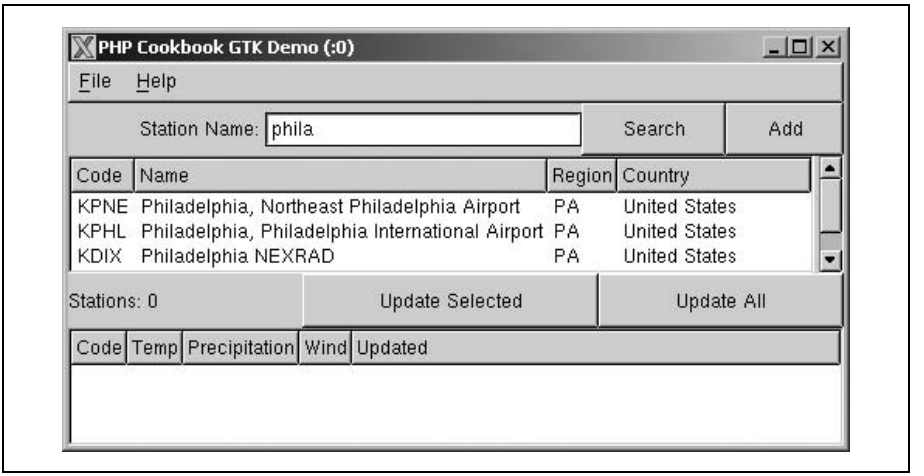

Рис. 20.4. *Поиск станций погоды*

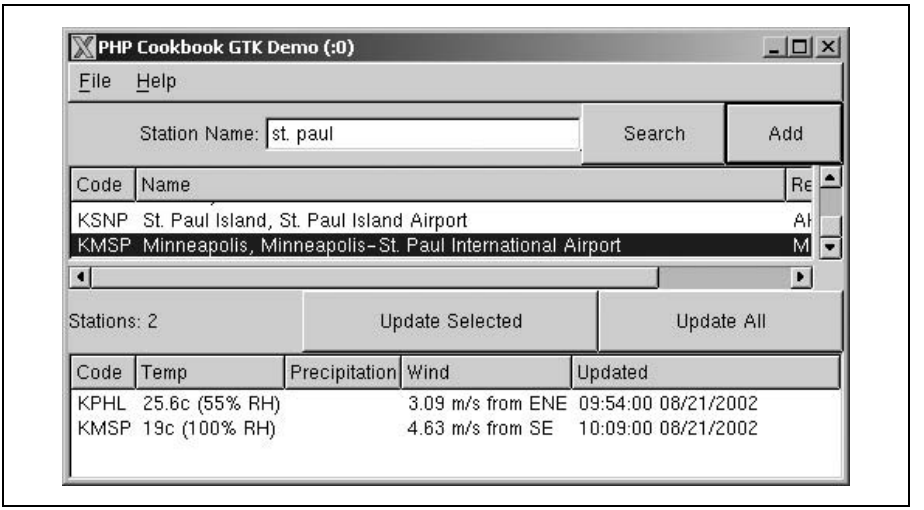

Рис. 20.5. *Добавленные станции погоды*

```
// Создаем главное окно и устанавливаем его название и размер
$window = &new GtkWindow();
$window->set_title('PHP Cookbook GTK Demo');
$window->set_default_size(500,200);

// Главным контейнером размещения для окна является VBox
$vbox = &new GtkVBox();
$window->add($vbox);

// Создаем объект GtkAccelGroup для хранения ускоряющих клавиш
$accelgroup = &new GtkAccelGroup();
$window->add_accel_group($accelgroup);
```

```
// Строим меню, начиная с GtkMenuBar. Аргументы функции pack_start()
// предотвращают расширение панели меню, если окно расширяется.
$menubar = &new GtkMenuBar();
$vbox->pack_start($menubar, false, false);

// Создаем меню «File» и его ускоряющую клавишу
$menu_file_item = &new GtkMenuItem('_File');
$menu_file_item_label = $menu_file_item->child;
$menu_file_item->add_accelerator('activate',$accelgroup,
                                 $menu_file_item_label->parse_uline('_File'),
                                  GDK_MOD1_MASK,0);
// Добавляем меню «File» в панель меню
$menubar->add($menu_file_item);

// Создаем подменю для опций в меню «File»
$menu_file_submenu = &new GtkMenu();
$menu_file_item->set_submenu($menu_file_submenu);

// Создаем опцию «Quit» внутри меню «File» и ее ускоритель
// GDK_MOD1_MASK означает, что ускорителем служит комбинация Alt-Q, а не Q
// GTK_ACCEL_VISIBLE означает, что ускоритель показывается в меню
$menu_file_choices_quit = &new GtkMenuItem('_Quit');
$menu_file_choices_quit_label = $menu_file_choices_quit->child;
$menu_file_choices_quit->add_accelerator('activate',$accelgroup,
    $menu_file_choices_quit_label->parse_uline('_Quit'),GDK_MOD1_MASK,
    GTK_ACCEL_VISIBLE);

// Добавляем опцию «File | Quit» в подменю «File»
$menu_file_submenu->append($menu_file_choices_quit);

// Создаем меню «Help» и его ускоряющую клавишу
$menu_help_item = &new GtkMenuItem('_Help');
$menu_help_item_label = $menu_help_item->child;
$menu_help_item->add_accelerator('activate',$accelgroup,
                                 $menu_help_item_label->parse_uline('_Help'),
                                  GDK_MOD1_MASK,0);
// Добавляем меню «Help» в панель меню
$menubar->add($menu_help_item);

// Создаем подменю для опций в меню «Help»
$menu_help_submenu = &new GtkMenu();
$menu_help_item->set_submenu($menu_help_submenu);

// Создаем опцию «About» в меню «Help» и ее ускоритель
$menu_help_choices_about = &new GtkMenuItem('_About');
$menu_help_choices_about_label = $menu_help_choices_about->child;
$menu_help_choices_about->add_accelerator('activate',$accelgroup,
    $menu_help_choices_about_label->parse_uline('_About'),GDK_MOD1_MASK,
    GTK_ACCEL_VISIBLE);

// Добавляем опцию «Help | About» в подменю «Help»
$menu_help_submenu->append($menu_help_choices_about);

// Схема размещения элементов управления поиска станций погоды в GtkTable
$table_1 = &new GtkTable(2,4);
```

```
$vbox->pack_start($table_1);

// Размещаем метку в первой строке слева
$label_sn = &new GtkLabel('Station Name: ');
$label_sn->set_alignment(1,0.5);
$table_1->attach($label_sn,0,1,0,1, GTK_FILL);

// Размещаем поле текстового ввода в середине первой строки
// Ускоритель позволяет для подтверждения нажать  «Return» в поле
$entry_sn = &new GtkEntry();
$entry_sn->add_accelerator('activate',$accelgroup,GDK_KEY_Return,0,0);
$table_1->attach($entry_sn,1,2,0,1, GTK_FILL);

// Размещаем окно с прокруткой во второй строке таблицы
$scrolledwindow_1 = &new GtkScrolledWindow();
$scrolledwindow_1->set_policy(GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);
$table_1->attach($scrolledwindow_1,0,4,1,2,
                 GTK_EXPAND | GTK_SHRINK | GTK_FILL,
                 GTK_EXPAND | GTK_SHRINK | GTK_FILL);

// Помещаем колоночный список в окно с прокруткой. Размещение списка внутри
// окна с прокруткой, вместо непосредственного размещения в GtkTable,
// позволяет окну не расти неограниченно для отображения всего списка
$clist_sn = &new GtkCList(4,array('Code','Name','Region','Country'));
$scrolledwindow_1->add($clist_sn);

// Устанавливаем автоматическое изменение размера столбцов в списке
for ($i = 0; $i < 4; $i++) { $clist_sn->set_column_auto_resize($i,true); }

// Добавляем кнопку «Search» в первую строку
$button_search =&new GtkButton('Search');
$table_1->attach($button_search,2,3,0,1, GTK_FILL);

// Добавляем кнопку «Add» в первую строку
$button_add = &new GtkButton('Add');
$table_1->attach($button_add,3,4,0,1, GTK_FILL);

// Схема размещения элементов управления,
// показывающих погодные условия, в другом GtkTable
$table_2 = &new GtkTable(2,3);
$vbox->pack_start($table_2);

// Добавляем метку, отображающую количество показанных станций
$label_st = &new GtkLabel('Stations: 0');
$label_st->set_alignment(0,0.5);
$table_2->attach($label_st,0,1,0,1, GTK_FILL);

// Добавляем кнопку для обновления одиночной станции
$button_update_sel = &new GtkButton('Update Selected');
$table_2->attach($button_update_sel,1,2,0,1, GTK_FILL);

// Добавляем кнопку для обновления всех станций
$button_update_all = &new GtkButton('Update All');
$table_2->attach($button_update_all,2,3,0,1, GTK_FILL);

// Добавляем колоночный список для хранения погодных условий на станциях
```

```
// Этот объединенный список также помещается в окно с прокруткой
$scrolledwindow_2 = &new GtkScrolledWindow();
$scrolledwindow_2->set_policy(GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);
$table_2->attach($scrolledwindow_2,0,3,1,2,
                 GTK_EXPAND | GTK_SHRINK | GTK_FILL,
                 GTK_EXPAND | GTK_SHRINK | GTK_FILL);
$clist_st = &new
GtkCList(5,array('Code','Temp','Precipitation','Wind','Updated'));
$scrolledwindow_2->add($clist_st);

// Устанавливаем автоматическое изменение размера столбцов в списке
for ($i = 0; $i < 5; $i++) { $clist_st->set_column_auto_resize($i,true); }

// Связываем сигналы с функциями обратного вызова

// Нажатие на кнопку «Search» или выбор Return в текстовом поле приводит
// к поиску станций погоды, чьи имена совпадают с введенным текстом
$button_search->connect('clicked','wx_searchByName',
                        $entry_sn,$clist_sn,$window);
$entry_sn->connect('activate','wx_searchByName',
                        $entry_sn,$clist_sn,$window);

// Нажатие на кнопку «Add» приводит к добавлению станции погоды
// в конце списка
$button_add->connect('clicked','cb_add_station',
                     $clist_sn,$clist_st,$label_st);

// Нажатие на кнопку «Update Selected» обновляет нижнюю часть
// колоночного списка для одиночной станции
$button_update_sel->connect('clicked','wx_update_report',
                            $clist_st,$label_st,'selected');
// Нажатие на кнопку «Update All» обновляет все станции
// внизу колоночного списка
$button_update_all->connect('clicked','wx_update_report',
                            $clist_st,$label_st,'all');

// Закрытие окна или выбор элемента меню «File | Quit» приводит
// к выходу из программы
$window->connect('destroy','cb_shutdown');
$menu_file_choices_quit->connect('activate','cb_shutdown');

// Выбор элемента меню «Help | About» показывает информационное окно
$menu_help_choices_about->connect('activate','cb_about_box',$window);

// Эти функции обратного вызова отслеживают текущую выбранную строку
// (если таковая имеется) в каждом колоночном списке
$clist_sn->connect('select-row','cb_clist_select_row');
$clist_sn->connect('unselect-row','cb_clist_unselect_row');
$clist_st->connect('select-row','cb_clist_select_row');
$clist_st->connect('unselect-row','cb_clist_unselect_row');

// Интерфейс установлен и необходимые сигналы привязаны к функциям
// обратного вызова. Время показывать окно и открывать
// цикл обработки сигналов GTK.
$window->show_all();
```

```
gtk::main();

/*
 * ФУНКЦИИ ОБРАТНОГО ВЫЗОВА  И ДРУГИЕ ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ
 */

// используем функцию searchByName() поверх SOAP для получения списка
// станций, имена которых соответствуют указанным условиям поиска
function wx_searchByName($button,$entry,$clist,$window) {
    // создаем экземпляр нового SOAP-клиента
    $sc = new SOAP_Client('http://live.capescience.com/ccx/GlobalWeather');

    $search_term = trim($entry->get_text());
    if ($search_term) {
        // вызываем удаленную функцию, если предоставлены условия поиска
        $res = $sc->call('searchByName',
                     array(new SOAP_Value('name','string',$search_term)),
                     'capeconnect:GlobalWeather:StationInfo',
                     'capeconnect:GlobalWeather:StationInfo#searchByName');

        // вызываем диалог ошибки, если SOAP-функция завершается неудачно
        if (PEAR::isError($res)) {
            error_dialog($res->getMessage(),$window);
            return false;
        }
        // вызываем диалог ошибки, если нет совпадений
        if (! is_array($res)) {
            error_dialog('No weather stations found.',$window);
            return false;
        }
        // добавляем каждую станцию и ее информацию в колоночный список,
        // вызывая функцию append() через вызовы функций freeze()
        // и thaw(), чтобы получить отображение всех данных одновременно
        $clist->freeze();
        $clist->clear();
        foreach ($res as $station) {
            $clist->append(array($station->icao,$station->name,
                                 $station->region,$station->country));
        }
        $clist->thaw();
    }
}

// используем функцию getWeatherReport поверх SOAP для получения
// погодных условий на конкретной станции
function wx_getWeatherReport($code) {
    $sc = new SOAP_Client('http://live.capescience.com/ccx/GlobalWeather');
    $res = $sc->call('getWeatherReport',
                array(new SOAP_Value('code','string',$code)),
                'capeconnect:GlobalWeather:GlobalWeather',
                'capeconnect:GlobalWeather:GlobalWeather#getWeatherReport');

    if (PEAR::isError($res)) {
        error_dialog($res->getMessage());
```

```
        return false;
    } else {
        return $res;
    }
}

// добавляем отчет о погоде в переменной $res в колоночный список $clist
// если переменная $row равна нулю, то отчет добавляется к списку
// если $row не равна нулю, то отчет замещает строку $row в списке
function wx_add_report($clist,$label,$res,$row = null) {

    // форматируем метку даты/времени
    $timestamp = str_replace('T',' ',$res->timestamp);
    $timestamp = str_replace('Z',' GMT',$timestamp);
    $timestamp = strftime('%H:%M:%S %m/%d/%Y',strtotime($timestamp));

    // форматируем информацию о ветре
    $wind = sprintf("%.2f m/s from %s",
                    $res->wind->prevailing_speed,
                    $res->wind->prevailing_direction->compass);

    $clist->freeze();
    if (! is_null($row)) {
        // замещаем информацию в строке с номером $row
        $clist->set_text($row,1,$res->temperature->string);
        $clist->set_text($row,2,$res->precipitation->string);
        $clist->set_text($row,3,$wind);
        $clist->set_text($row,4,$timestamp);
    } else {
        // добавляем информацию в конец колоночного списка
        $clist->append(array($res->station->icao,
                             $res->temperature->string,
                             $res->precipitation->string,
                             $wind,
                             $timestamp));

        // обновляем внутренний счетчик строк колоночного списка
        $rows = 1 + $clist->get_data('rows');
        $clist->set_data('rows',$rows);
        // обновляем метку, показывающую счетчик станций
        $label->set_text("Stations: $rows");
    }
    $clist->thaw();
}

// обновляем условия одной станции или всех станций
// в зависимости от значения $mode
function wx_update_report($button,$clist,$label,$mode) {
    switch ($mode) {
    case 'selected':

        // если есть выбранная строка
        $selected_row = $clist->get_data('selected_row');
        if (($selected_row >= 0) && (! is_null($selected_row))) {
```

```
            $code = $clist->get_text($selected_row,0);

            // берем отчет и обновляем колоночный список
            if ($res = wx_getWeatherReport($code)) {
                wx_add_report($clist,$label,$res,$selected_row);
            }
        }
        break;
    case 'all':
        // для каждой строки колоночного списка
        for ($i = 0, $j = $clist->get_data('rows'); $i < $j; $i++) {
            // берем отчет и обновляем список
            if ($res = wx_getWeatherReport($clist->get_text($i,0))) {
                wx_add_report($clist,$label,$res,$i);
            }
        }
        break;
    }
}

// добавляем станцию в конец списка отчетов о погоде
function cb_add_station($button,$clist,$clist_2,$label) {
    $selected_row = $clist->get_data('selected_row');
    // если есть выбранная строка в начале списка станций
    if ($selected_row >= 0) {
        $code = $clist->get_text($selected_row,0);
        // получаем отчет для этой станции
        if ($res = wx_getWeatherReport($code)) {
            // находим строку, если этот код уже находится в списке
            $row = null;
            for ($i = 0, $j = $clist_2->get_data('rows'); $i < $j; $i++) {
                if ($clist_2->get_text($i,0) == $code) {
                    $row = $i;
                }
            }
            // добавляем станцию и ее отчет внизу списка
            // отчетов (или обновляем существующую строку)
            wx_add_report($clist_2,$label,$res,$row);
        }
    }
}

// обновляем внутреннее значение выбранной строки колоночного списка,
// когда строка выбирается
function cb_clist_select_row($clist,$row,$col,$e) {
    $clist->set_data('selected_row',$row);
}

// удаляем внутреннее значение выбранной строки колоночного списка,
// когда выбор строки снимается
function cb_clist_unselect_row($clist) {
    $clist->set_data('selected_row',-1);
}
```

```
// показываем «About Box»
function cb_about_box($menu_item,$window) {
    $about_box = &new GtkDialog();
    $vbox = $about_box->vbox;
    $action_area = $about_box->action_area;
    $about_box->set_title('About');
    $label = &new GtkLabel("This is the PHP Cookbook PHP-GTK Demo.");
    $button = &new GtkButton('OK');
    $button->connect('clicked','cb_dialog_destroy',$about_box);
    $vbox->pack_start($label);
    $action_area->pack_start($button);
    $about_box->set_modal(true);
    $about_box->set_transient_for($window);
    $about_box->show_all();
}

// показываем окно диалога ошибки
function error_dialog($msg,$window) {
    $dialog = &new GtkDialog();
    $vbox = $dialog->vbox;
    $action_area = $dialog->action_area;
    $dialog->set_title('Error');
    $label = &new GtkLabel("Error: $msg");
    $button = &new GtkButton('OK');
    $button->connect('clicked','cb_dialog_destroy',$dialog);
    $vbox->pack_start($label);
    $action_area->pack_start($button);
    $dialog->set_modal(true);
    $dialog->set_transient_for($window);
    $dialog->show_all();
}

// закрываем окно диалога
function cb_dialog_destroy($button,$dialog) {
    $dialog->destroy();
}

// выходим из главной программы
function cb_shutdown() { gtk::main_quit(); }
```

21

PEAR

21.0. Введение

PEAR – это и расширение PHP, и одновременно набор готовых программ. PEAR представляет собой коллекцию разнообразных классов с открытым кодом, работающих совместно. PEAR-классы можно использовать для генерации HTML, создания SOAP-запросов, отправки MIME-почты и решения множества других распространенных задач. Кроме того, груша (pear) – очень вкусный фрукт.

Основную информацию о PEAR можно получить из руководства; самые последние пакеты PEAR всегда можно найти на *http://pear.php.net*. Обзоры еженедельных событий в мире PEAR регулярно публикуются на *http://pear.php.net/weeklynews.php*.

Лишь небольшая часть пакетов ядра PEAR тесно связана с основной версией PHP. Тем не менее эта часть PEAR – самодостаточная программа с подходящим названием *pear*, которая облегчает загрузку и инсталляцию дополнительных пакетов PEAR. Эта программа известна еще как менеджер пакетов PEAR. Работа с ним описана в рецепте 21.1.

Пакеты PEAR делятся на две основные части. Одна из них – это основные классы PHP (PHP Foundation Classes) – объектно-ориентированный код высокого качества, написанный на PHP, который может быть использован в среде разработки на любой платформе и на любом веб-сервере. Другая – это PECL, или библиотека дополнительных программ PHP (PHP Extension Code Library). PECL, произносимая как «пикл», представляет собой набор расширений PHP, написанных на языке C. Эти расширения похожи на расширения, поставляемые с основной версией PHP, но носят более специальный характер – реализуют, например, интерфейс к мультимедиа плееру XMMS или графической библиотеке ImageMagick.

Кроме того, благодаря менеджеру пакетов PEAR пользователи получают возможность применять инфраструктуру управления PEAR-классами в собственных проектах. Создавая пакеты, поддерживающие

формат PEAR, они смогут посредством *pear* реализовать загрузку и инсталляцию файлов с веб-сайта вашего проекта.

В этой главе объясняется, как найти необходимый пакет PEAR и установить его на своей машине. Поскольку в PEAR много классов, необходимо также иметь удобный способ их просмотра. Рецепт 21.2 посвящен различным методам нахождения пакетов PEAR, в рецепте 21.3 показано, как просмотреть более подробную информацию о пакете, если его имя уже найдено.

Определив, какой класс вам нужен, запустите *pear*, чтобы переправить класс на свою машину и установите его в нужном месте на сервере. Установка пакетов PEAR и расширений PECL рассмотрены в рецептах 21.4 и 21.5 соответственно. В рецепте 21.6 показано, как определить, есть ли доступные обновления для пакетов вашей машины и как инсталлировать последние версии. О том, как удалить пакет, расскажет рецепт 21.7.

Наконец, в рецепте 21.8 рассказывается, как разработчики, использующие PEAR, могут создавать новые классы в соответствии со стандартами программирования PEAR и как документировать классы с помощью PHPDoc.

PHP 4.3 включает первую стабильную версию PEAR. Более ранние версии PHP связаны с версиями PEAR до PEAR 1.0, но работа *pear* и других ключевых пакетов не гарантировалась, поскольку все это существовало в статусе бета-версии. Если возникли проблемы с применением PEAR, то необходимо удалить все старые файлы, которые могут пересекаться с файлами рабочей версии. Это относится и к самому приложению *pear*; оно не всегда может обновить себя до последней версии.

Если вы не можете обновить PHP до версии 4.3, а поставить свежую копию PEAR на систему надо, то выполните следующее:

```
% lynx -source http://go-pear.org | php -q
Welcome to go-pear!
(Добро пожаловать в go-pear!)

Go-pear will install the 'pear' command and all the files needed by
it.  This command is your tool for PEAR installation and maintenance.
(Go-pear установит программу 'pear' и все файлы, необходимые для нее.
Эта программа - ваш инструмент для инсталляции и поддержки PEAR.)

Go-pear also lets you download and install the PEAR packages bundled
with PHP: DB, Net_Socket, Net_SMTP, Mail, XML_Parser.
(Go-pear также позволяет загружать и инсталлировать пакеты PEAR, с PHP: DB,
связанные Net_Socket, Net_SMTP, Mail, XML_Parser.)

If you wish to abort, press Control-C now, or press Enter to continue:
(Если вы хотите выйти, нажмите Control-C, или нажмите Enter для продолжения:)
```

В результате этих действий нужный PHP-сценарий будет загружен с веб-сайта PEAR и передан на выполнение интерпретатору PHP. Эта

программа загружает все файлы, необходимые для запуска *pear*, и подготавливает его к выполнению.

В некоторых системах UNIX может потребоваться запустить *links* вместо *lynx*. Если у вас инсталлирована версия PHP для командной строки, то удалите флаг `-q` рядом с вызовом PHP; версия CLI автоматически подавляет вывод HTTP-заголовков. Если возникает ощущение, что *go-pear* зависает, установите параметр `output_buffering` в `off` в файле *php.ini*.

Инсталляция в Windows происходит в два этапа:

```
C:\> php-cli -r 'readfile("http://go-pear.org");' > go-pear
c:\> php-cli go-pear
```

Сценарий *go-pear* требует PHP версии 4.1 или выше. В случае инсталляции в Windows *php-cli* представляет собой версию PHP для командной строки.

PHP инсталлирует PEAR по умолчанию, поэтому если запускается версия PHP 4.3, то можно использовать PEAR без каких-либо дополнительных установок.[1] Новый PEAR инсталлирует *pear* в тот же самый каталог, что и *php*, и помещает пакеты PEAR в *prefix/lib/php*.[2] Для установки PEAR в другой каталог добавьте параметр `--with-pear=DIR` при настройке PHP.

Установленный пакет PEAR используется в сценариях с помощью вызова `require`. Ниже показано, как включить пакет `Net_Dig`:

```
require 'Net/Dig.php';
```

Символ подчеркивания, содержащийся в имени пакета, надо заменить символом косой черты и добавить к имени *.php*.

Некоторые пакеты могут потребовать включения нескольких классов, например SOAP, поэтому вместо вызова *SOAP.php* надо подключать *SOAP/Client.php* или *SOAP/Server.php*. Ознакомьтесь с документацией, чтобы узнать, не нуждается ли конкретный пакет в таком нестандартном включении классов.

Поскольку пакеты PEAR подключаются как обычные файлы PHP, убедитесь, что каталог, содержащий PEAR-классы, входит в список каталогов, указанных в `include_path`. Если это будет не так, то вызовы `include` и `require` не смогут найти PEAR-классы.

Инструкции и примеры применения отдельных PEAR-классов можно найти в руководстве по PEAR на *http://pear.php.net/manual/en/packages.php* или просмотрите внимательно начальный раздел файлов с пакетами PHP. Пример PEAR-класса с полной демонстрацией его воз-

---

[1] Если вы запретите сборку версии PHP для командной строки с помощью параметра `--disable-cli`, то PHP не инсталлирует PEAR.

[2] Возможно, это */usr/local/lib/php*.

можностей в действии можно увидеть в рецепте 10.3, в котором обсуждается библиотека базы данных PEAR.

21.1. Работа с менеджером пакетов PEAR

Задача

Необходимо использовать менеджер пакетов PEAR, *pear*. Он позволяет инсталлировать пакеты, а также обновлять и получать информацию об уже установленных пакетах PEAR.

Решение

Чтобы выполнить команду с помощью менеджера пакетов PEAR, введите имя этой команды в качестве первого аргумента командной строки:

```
% pear command
```

Обсуждение

Ниже показано, как получить список всех инсталлированных пакетов с помощью команды `list`:[1]

```
% pear list
Installed packages:
===================
+-----------------+----------+--------+
| Package         | Version  | State  |
| Archive_Tar     | 0.9      | stable |
| Console_Getopt  | 0.11     | beta   |
| DB              | 1.3      | stable |
| HTTP            | 1.2      | stable |
| Mail            | 1.0.1    | stable |
| Mail_Mime       | 1.2.1    | stable |
| Net_SMTP        | 1.0      | stable |
| Net_Socket      | 1.0.1    | stable |
| Net_URL         | 1.0.4    | stable |
| PEAR            | 0.91-dev | beta   |
| XML_Parser      | 1.0      | stable |
| XML_RPC         | 1.0.3    | stable |
+-----------------+----------+--------+
```

Для получения перечня всех допустимых команд PEAR применяется команда `list-commands`. Многие команды имеют и сокращенные имена, например `list` сокращенно просто `l`. Чаще всего эти имена представляют первые буквы имени команды. Список часто применяемых команд приводится в табл. 21.1.

[1] В ранних версиях *pear* это была команда `list-installed`.

Таблица 21.1. Команды менеджера пакетов PEAR

Имя команды	Сокращение	Описание
Install	I	Загружает и инсталлирует пакеты
Upgrade	Up	Обновляет инсталлированные пакеты
Uninstall	Un	Удаляет инсталлированные пакеты
List	L	Составляет список инсталлированных пакетов
list-upgrades	Lu	Составляет список доступных обновлений для инсталлированных пакетов
Search	None	Ищет пакеты

Программа *pear* содержит команды как для использования, так и для разработки PEAR-классов; конечно, все команды могут вам и не потребоваться. Например, команда package создает новый пакет PEAR. Если вы только используете пакеты, разработанные другими людьми, то спокойно можете игнорировать эту команду.

Как для любой другой программы, необходимо иметь право на запуск *pear*. Если вы можете запустить *pear* от имени пользователя root, но не можете запустить ее от имени конкретного пользователя, нужно убедиться в том, что для программы установлен соответствующий бит выполнения от имени группы, или право на выполнение для любого пользователя. В некоторых ситуациях *pear* создает lock-файл в каталоге, содержащем файлы PEAR. Необходимо обладать правом на запись в этот каталог, чтобы файлы с именем *.lock* могли успешно создаваться.

Чтобы определить, где находятся ваши пакеты PEAR, выполните команду config-get php_dir. Можно проверить значение include_path с помощью вызова ini_get('include_path') из PHP или напрямую просмотреть файл *php.ini*. Если у вас нет собственного альтернативного *php.ini* и вы ограничены условиями виртуального хостинга, то добавьте соответствующий каталог к значению include_path в начале своего сценария, перед тем как подключить файл. Дополнительная информация по установке переменных конфигурации из PHP содержится в рецепте 8.23.

Если вы находитесь за прокси-сервером HTTP, сконфигурируйте PEAR для его использования при помощи команды:

```
% pear config-set http_proxy proxy.example.com:8080
```

Настраивать PEAR можно с помощью менеджера пакетов:

```
% pear set-config setting value
```

Здесь setting — это имя модифицируемого параметра, а value — его новое значение. Посмотреть все текущие настройки позволяет команда config-show:

```
% pear config-show
Configuration:
```

```
==============
+---------------------+--------------+-------------------------------+
| PEAR executables    | bin_dir      | /usr/local/bin                |
| directory           |              |                               |
| PEAR documentation  | doc_dir      | /usr/local/lib/php/docs       |
| directory           |              |                               |
| PHP extension       | ext_dir      | /usr/local/lib/php/           |
|                     |              | /extensions/no-de             |
| directory           |              | bug-non-zts-20020429          |
| PEAR directory      | php_dir      | /usr/local/lib/php            |
| PEAR data directory | data_dir     | /usr/local/lib/php/data       |
| PEAR test directory | test_dir     | /usr/local/lib/php/tests      |
| HTTP Proxy Server   | http_proxy   | <not set>                     |
| Address             |              |                               |
| PEAR server         | master_server | pear.php.net                  |
| PEAR password (for  | password     | <not set>                     |
| maintainers)        |              |                               |
| PEAR username (for  | username     | <not set>                     |
| maintainers)        |              |                               |
| Preferred Package   | preferred_state | stable                        |
| State               |              |                               |
| UNIX file mask      | umask        | 18                            |
| Debug Log Level     | verbose      | 1                             |
+---------------------+--------------+-------------------------------+
```

Для того чтобы получить краткое описание каждого параметра конфигурации, применяется команда `config-help`.

21.2. Нахождение пакетов PEAR

Задача

Требуется получить перечень пакетов PEAR, а из этого перечня – дополнительную информацию о каждом пакете, и решить, надо ли вам инсталлировать тот или иной пакет.

Решение

Просмотрите пакеты на *http://pear.php.net/packages.php* или на *http://pear.php.net/package-search.php*. Для этого применяется *pear*-команда `remote-list`, позволяющая просмотреть список пакетов, или команда `search`, выполняющая поиск пакетов.

Обсуждение

Существует несколько способов просмотреть пакеты PEAR. Во-первых, для просмотра списков в виде каталогов обратитесь по адресу *http://pear.php.net/packages.php*. Отсюда можно начать углубленный поиск в каждой отдельной категории пакетов PEAR.

Есть и альтернатива – можно поискать в списках на *http://pear.php.net/package-search.php*. На странице поиска поддерживается поиск по имени пакета, автору, категории и дате выпуска.

Можно запросить у менеджера PEAR список пакетов – это делается с помощью команды `remote-list`:

```
% pear remote-list
Available packages:
===================
+----------------------+---------+
| Package              | Version |
| Archive_Tar          | 0.9     |
| Auth                 | 1.0.2   |
...
| XML_Transformer      | 0.3     |
| XML_Tree             | 1.1     |
+----------------------+---------+
```

Краткая форма команды `remote-list` это `rl`.

Для поиска имен пакетов из командной строки применяется команда `search`:

```
% pear search auth
Matched packages:
=================
+-----------+--------+-------+-----------------------------------+
| Package   | Latest | Local |                                   |
| Auth      | 1.0.2  | 1.0.2 | Creating an authentication system. |
| Auth_HTTP | 1.0.1  | 1.0.1 | HTTP authentication for PHP       |
+-----------+--------+-------+-----------------------------------+
```

Она выполняет поиск имен пакетов, чувствительный к регистру, и возвращает имя пакета, номер последней версии, установленную версию (если таковая есть) и краткое описание пакета.

См. также

Рецепт 21.3 о поиске дополнительной информации о пакете.

21.3. Поиск информации о пакете

Задача

Необходимо собрать информацию о пакете (например, что он делает, кем поддерживается, какая версия у вас установлена и под какой лицензией он выпущен).

Решение

Если пакет установлен на вашей машине, то выполните команду `info` менеджера пакетов PEAR:

```
% pear info Net_URL
```

В противном случае выполните команду `remote-info`:

```
% pear remote-info SOAP
```

Можно также посмотреть домашнюю страничку пакетов на *http://pear.php.net*.

Обсуждение

Команда `info` предоставляет суммарную информацию о пакете:

```
% pear info Net_URL
About Net_URL-1.0.4
===================
+-----------------+----------------------------------------+
| Package         | Net_URL                                |
| Summary         | Easy parsing of Urls                   |
| Description     | Provides easy parsing of URLs and their |
|                 | constituent parts.                     |
| Maintainers     | Richard heyes <richard@php.net> (lead)  |
| Version         | 1.0.4                                  |
| Release Date    | 2002-07-27                             |
| Release License | BSD                                    |
| Release State   | stable                                 |
| Release Notes   | License change                         |
| Last Modified   | 2002-08-23                             |
+-----------------+----------------------------------------+
```

Если пакет не инсталлирован у вас, запросите его описание на удаленном сервере:

```
% pear remote-info Net_URL
Package details:
================
+-------------+----------------------------------------+
| Latest      | 1.0.4                                  |
| Installed   | 1.0.4                                  |
| Package     | Net_URL                                |
| License     | BSD                                    |
| Category    | Networking                             |
| Summary     | Easy parsing of Urls                   |
| Description | Provides easy parsing of URLs and their |
|             | constituent parts.                     |
+-------------+----------------------------------------+
```

Этот запрос показывает несколько отличный набор информации. Он не включает дату выпуска, но содержит основную категорию PEAR и номер последней версии пакета.

Типичная «домашняя страничка» пакетов предоставляет более полный обзор и содержит ссылки на более ранние выпуски, журнал изменений и позволяет получить доступ на просмотр к репозиторию CVS. Здесь же можно посмотреть статистику загрузки пакетов. На рис. 21.1 показана традиционная страница с информацией о пакете.

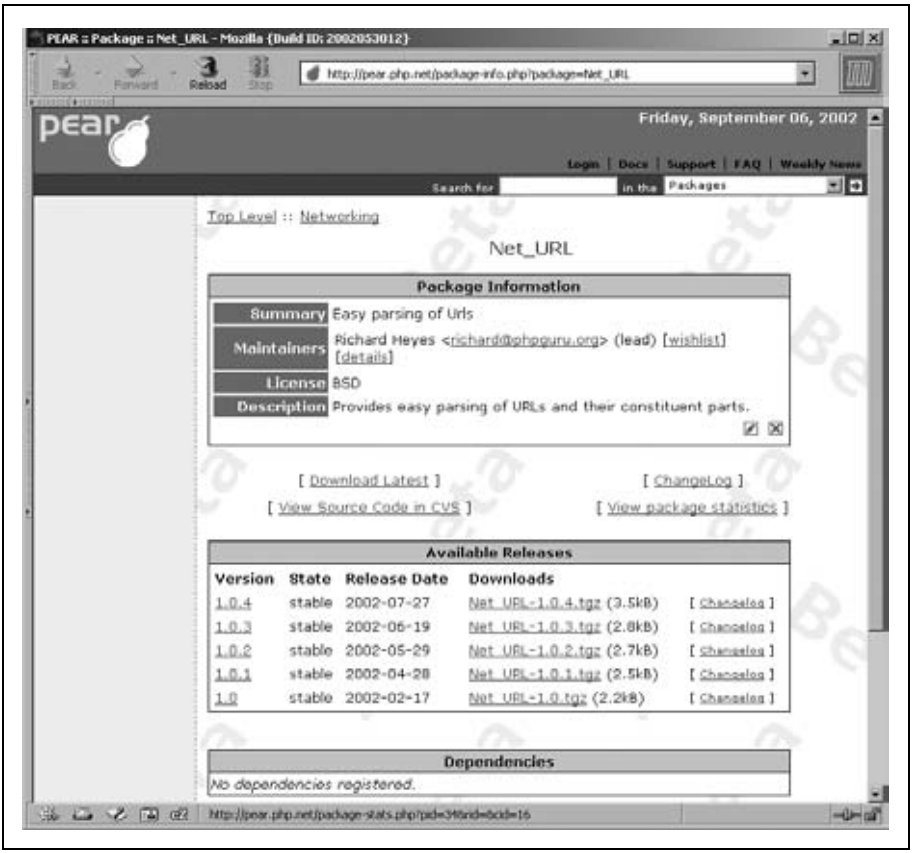

Рис. 21.1. *Страница с информацией о пакете Net_URL на веб-сайте PEAR*

См. также

Рецепт 21.2 о поиске пакетов.

21.4. Установка пакетов PEAR

Задача

Необходимо инсталлировать пакет PEAR.

Решение

Загрузите и установите пакет с вашего сервера PEAR с помощью менеджера пакетов PEAR:

```
% pear install Package_Name
```

Установить пакет можно из любого другого места в Интернете, непосредственно указав сетевой адрес:

```
% pear install http://pear.example.com/Package_Name-1.0.tgz
```

Для того чтобы установить пакет из локальной копии, которая находится на компьютере в виде архива, надо выполнить команду:

```
% pear install Package_Name-1.0.tgz
```

Обсуждение

Для инсталляции пакетов PEAR необходимы права записи в каталог установки пакетов; по умолчанию это */usr/local/lib/php/*.

Можно запросить несколько пакетов одновременно:

```
% pear install HTML_Common HTML_Javascript
downloading HTML_Common-1.0.tgz ...
...done: 2,959 bytes
install ok: HTML_Common 1.0
downloading HTML_Javascript-1.0.0.tgz ...
...done: 4,141 bytes
install ok: HTML_Javascript 1.0.0
```

При инсталлировании пакета PEAR проверяет, что у вас есть все необходимые функции PHP и пакеты PEAR, от которых зависит новый пакет. Если проверка не проходит, то PEAR выведет сообщение о найденных зависимостях:

```
% pear install HTML_Table
downloading HTML_Table-1.1.tgz ...
...done: 5,168 bytes

requires package `HTML_Common' >= 1.0
HTML_Table: dependencies failed
```

Чтобы решить проблему, сначала загрузите и инсталлируйте недостающие пакеты. Если нужно проигнорировать эти зависимости, то форсируйте инсталляцию с помощью опции `-n` или `--nodeps`. Дополнительные, недостающие пакеты, можно будет установить позже.

См. также

Рецепт 21.5 об инсталляции пакетов PECL; дополнительную информацию об обновлении существующих пакетов в рецепте 21.6; рецепт 21.7 о деинсталляции пакета.

21.5. Установка пакетов PECL

Задача

Необходимо инсталлировать пакет PECL; в результате инсталляции будет собрано расширение PHP, написанное на языке C, используемое затем как динамическая библиотека PHP.

Решение

Убедитесь, что у вас есть все необходимые библиотеки расширений, а затем выполните команду `install` менеджера пакетов PEAR:

```
% pear install xmms
```

Для того чтобы использовать расширение из PHP, надо загрузить его с помощью функции `dl()`:

```
dl('xmms.so');
```

Обсуждение

Внешне процесс инсталляции пакетов PECL схож с процессом установки кода пакетов PEAR, написанных на PHP. Однако задачи, выполняемые при этом неявно, значительно отличаются. Поскольку расширения PECL написаны на языке C, менеджер пакетов должен скомпилировать и сконфигурировать пакет для работы с уже инсталлированной версией PHP. Поэтому в настоящее время можно собрать пакеты PECL на UNIX-машинах и на тех Windows-машинах, где установлена среда разработки MSDev.

В отличие от пакетов PEAR, разработанных на PHP, расширения PECL сами не информируют пользователя об отсутствии той или иной библиотеки, необходимой для установки расширения. Ответственность за предварительную инсталляцию таких файлов лежит на пользователе. Если собрать расширение PECL почему-либо не удается, то надо почитать файл *README* и документацию, входящую в состав пакета. Менеджер пакетов инсталлирует файлы документации в каталог *docs* иерархии PEAR.

При инсталляции расширения PECL менеджер пакетов PEAR загружает файл, распаковывает его, запускает программу *phpize*, чтобы сконфигурировать расширение под текущую версию PHP, установленную на вашей машине, и только затем собирает и инсталлирует расширение. В процессе установки он может запросить вас о местоположении необходимых библиотек:

```
% pear install xmms
downloading xmms-0.2.tgz ...
...done: 11,968 bytes
4 source files, building
running: phpize
PHP Api Version        : 20020307
Zend Module Api No     : 20020429
Zend Extension Api No  : 20020731
Xmms library install dir? [autodetect] :
building in /var/tmp/pear-build-adam/xmms-0.2
running: /tmp/pearKIv63P/xmms-0.2/configure --with-xmms
running: make
xmms.so copied to /tmp/pearKIv63P/xmms-0.2/xmms.so
install ok: xmms 0.2
```

Если библиотеки в вашей системе расположены стандартным образом, то просто нажмите на клавишу Return. Это приведет к выбору опции autodetect. Теперь PHP отыщет библиотеки и подключит их. При стандартном расположении нет необходимости явно вводить путь, как это показано выше на примере библиотеки xmms.

Расширения PECL располагаются не в тех местах, где хранятся остальные, не-PECL пакеты. Для того чтобы запускать и использовать *pear*, необходимо иметь право записи в каталог *расширений* PHP. Некоторые пакеты, такие как xmms, инсталлируют файлы в тот же каталог, в котором находится и интерпретатор PHP. Вследствие этого может потребоваться инсталлировать эти пакеты от имени того же пользователя, под которым был установлен PHP. Кроме того, надо проверить права выполнения для этих файлов. Большинство файлов PEAR неисполняемые, поэтому ваша маска (umask) может не установить необходимый набор прав доступа для исполняемых файлов.

См. также

Рецепт 21.4 об инсталляции пакетов PEAR; дополнительную информацию об обновлении существующих пакетов в рецепте 21.6; рецепт 21.7 о деинсталляции пакета.

21.6. Обновление пакетов PEAR

Задача

Необходимо установить последнюю версию пакета в системе, чтобы получить дополнительную функциональность и устранить ошибки.

Решение

Определите наличие доступных обновлений, а затем прикажите программе *pear* обновить требуемый пакет:

```
% pear list-upgrades
% pear upgrade Package_Name
```

Обсуждение

Менеджер пакетов PEAR делает установку новой версии пакета простой задачей. Если известно, что определенный пакет устарел, то его можно сразу обновить. Можно также периодически проверять, не появились ли новые версии.

Чтобы сделать это, выполните команду list-upgrades, которая выводит таблицу, показывающую имена пакетов, номер новой версии и объем загрузки:

```
% pear list-upgrades
Available Upgrades (stable):
============================
```

```
+-------------+---------+--------+
| Package     | Version | Size   |
| Archive_Tar | 0.9     | 8.9kB  |
| Auth        | 1.0.2   | 8.8kB  |
| Auth_HTTP   | 1.0.1   | 1.7kB  |
| DB          | 1.3     | 58kB   |
| HTTP        | 1.1     | 2.9kB  |
| Mail        | 1.0.1   | 11.6kB |
| Mail_Mime   | 1.2.1   | 15.0kB |
| Net_Ping    | 1.0.1   | 2.1kB  |
| Net_SMTP    | 1.0     | 2.8kB  |
| Net_Socket  | 1.0.1   | 3.5kB  |
| PEAR        | 0.9     | 40kB   |
| XML_Parser  | 1.0     | 4.8kB  |
| XML_RPC     | 1.0.3   | 11.9kB |
| XML_RSS     | 0.9.1   | 3.1kB  |
| XML_Tree    | 1.1     | 4.7kB  |
+-------------+---------+--------+
```

Если ваши версии пакетов не устарели, *pear* печатает:

```
No upgrades available
(доступных обновлений нет)
```

Для обновления определенного пакета применяется команда `upgrade`. Например:

```
% pear upgrade DB
downloading DB-1.3.tgz ...
...done: 59,332 bytes
```

Сокращенный вариант команды `list-upgrades` — `lu`, а команды `upgrade` — `up`.

У PEAR есть также свой перечень RSS-рассылок новых пакетов, доступный на *http://pear.php.net/rss.php*.

См. также

Информацию об установке пакетов PEAR и PECL в рецептах 21.4 и 21.5; рецепт 21.7 о деинсталляции пакета; дополнительную информацию об анализе RSS-рассылок в рецепте 12.11.

21.7. Удаление пакетов PEAR

Задача

Необходимо удалить пакет PEAR из системы.

Решение

Команда `uninstall` приказывает менеджеру пакетов PEAR удалить пакеты:

```
% pear uninstall HTML_Common
uninstall HTML_Common ok
```

Обсуждение

Деинсталляция пакета полностью удаляет его из системы. Если впоследствии понадобится реинсталлировать его, то придется начать заново, как будто пакет никогда не был установлен. PEAR не предупреждает дополнительно о том, что вы пытаетесь удалить пакет, который зависит от другого пакета, поэтому будьте крайне осторожны и внимательны при деинсталляции.

В случае применения `uninstall` не существует способа выполнить откат обновления к более ранней версии пакета. Кроме того, PEAR плохо реагирует на попытки установить более раннюю версию поверх более новой. Для того чтобы заставить PEAR переписать более новую версию, применяется команда `install -f`, или `install --force`:

```
% pear install --force Net_URL
downloading Net_URL-1.0.4.tgz ...
...done: 3,540 bytes
install ok: Net_URL 1.0.4
```

Сокращенный вариант команды `uninstall` — `un`.

См. также

Рецепты 21.4 и Recipe 21.5 об установке пакетов PEAR и PECL.

21.8. Документирование классов с помощью PHPDoc

Задача

Необходимо иметь возможность интегрировать документацию с программой.

Решение

Для этого предназначен PHPDoc, позволяющий PEAR составить точный перечень ваших классов; таким образом, инструментарий PHPDoc можно применять для автоматической генерации API-документации, написанной на HTML и XML.

Синтаксис PHPDoc основан на Javadoc. Допустимо применение следующих тегов: `@access`, `@author`, `@package`, `@param`, `@return`, `@since`, `@var` и `@version`.

Теперь утилиту PEAR PHPDoc можно применять для создания документации.

Обсуждение

PHPDoc имеет специальный построчный стиль документирования. Если отформатировать комментарии подобным образом, сценарий

PHPDoc сможет проанализировать программу и не только сгенерировать информацию о том, какие параметры принимает функция и какого типа переменную она возвращает, но и связать комментарии и другие полезные данные с объектами, функциями и переменными.

Комментарии PHPDoc основаны на тех же самых соглашениях о способах форматирования и именования, что приняты в Javadoc. Поэтому блок комментариев, предназначенный для обработки PHPDoc, обозначается посредством традиционного комментария, принятого в C, но только за открывающей косой чертой должны располагаться две звездочки:

```
/**
* Это блок комментариев PHPDoc
*/
```

Внутри такого блока определенные ключевые слова приобретают дополнительное специальное значение. Все эти ключевые слова начинаются с символа «@», или с «собаки». В табл. 21.2 приведены ключевые слова и их смысл для системы документирования.

Таблица 21.2. Ключевые слова PHPDoc

Ключевое слово	Смысл
@access	Метод доступа: public или private
@author	Автор пакета
@package	Имя пакета
@param	Параметр функции
@return	Возвращаемое функцией значение
@see	См. также ссылку
@since	Начальная версия PHP
@var	Переменная объекта
@version	Номер версии пакета

Наиболее полный пример выглядит так:

```
/**
* Example_Class - это простой класс для демонстрации PHPDoc
*
* Example_Class  не содержит полезного кода, а служит
* для наглядной демонстрации того, как применяются
* и используются различные служебные конструкции PHPDoc.
*
* Пример применения:
* if (Example_Class::example()) {
*    print "I am an example.";
* }
*
* @package  Example
```

```
 * @author   David Sklar <david@example.com>
 * @author   Adam Trachtenberg <adam@example.com>
 * @version  $Revision: 1.30 $
 * @access   public
 * @see      http://www.example.com/pear
 */
class Example extends PEAR
{
    /**
    * returns the sample data
    *
    * @param  string  $sample the sample data
    * @return array   all of the exciting sample options
    * @access private
    */
    function _sampleMe($sample)
    {
```

Любой текст, следующий за ключевым словом, трактуется как значение, присвоенное этому слову. Поэтому в данном примере значением `@package` является «Example». В зависимости от ситуации допускается наличие более одного экземпляра для ключевого слова. Например, вполне законно иметь несколько ключевых слов `@param`, но наличие нескольких слов `@return` недопустимо.

PHPDoc и веб-сайт PEAR используют эту информацию для создания гиперссылок, поэтому важно руководствоваться согласованной схемой именования, иначе перекрестные ссылки будут работать неправильно.

Чтобы собрать PHPDoc и потом работать с ней, сначала надо установить пакет PEAR PHPDoc. Внутри этого пакета находится программа *phpdoc*. Запустите ее из командной строки с флагом `-s` для передачи каталога исходных файлов. По умолчанию документация создается в каталоге */usr/local/doc/pear/*, поэтому убедитесь, что программа *phpdoc* имеет право записи в этот каталог, или измените место назначения с помощью аргумента `-d`.

Чтобы окончательно изменить значения, принятые по умолчанию, отредактируйте значения в начале сценария. Вызов с параметром `-h` выдаст список всех возможных параметров командной строки.

PHPDoc не очень эффективен и быстр, поэтому будьте терпеливы. Создание документации может занять время, зависящее во многом от размера ваших файлов. В настоящее время разрабатывается более быстрая версия программы.

См. также

Стандарты программирования PEAR на *http://pear.php.net/manual/en/standards.php*; PHPDoc на *http://pear.php.net/package-info.php?package=PHPDoc*.

Алфавитный указатель

Специальные символы

" (двойные кавычки)
интерполяция переменных в строке в двойных кавычках, 484
$ (знак доллара)
$_ в именах суперглобальных массивов, 249
привязка шаблона к концу строки, 389
строки в двойных кавычках, 24
% (знак процента)
групповой символ в SQL, 296
перед символами форматирования strftime(), функция, 76
& (амперсанд)
&, оператор (логическое И), 223
перед именами параметров функции, передача параметров по ссылке, 164
перед именем функции, возвращение значений по ссылке, 169
преобразование в элемент HTML, 212, 336
разделитель аргументов URL, 212
символ-заместитель в запросах к базам данных, 293
() (круглые скобки)
группировка символов и выделение в шаблонах, 389
комментарии, внедренные в адреса электронной почты, 399
* (звездочка), 296
метасимвол регулярного выражения, 388
+ (знак плюс)
метасимвол регулярного выражения, 388
оператор +, соединение массивов, 115
, (запятая)
данные разделенные, анализ, 36
массив элементов, разделение, 118
. (точка)
в имени поля HTML, преобразование в переменную PHP, 268
воссоздание полного имени файла из составляющих, 555
метасимвол регулярного выражения, 388
оператор конкатенации строк, 34
/ (косая черта)
/** и */ в комментариях PHPDoc, 621
воссоздание полного имени файла из составляющих, 555
разделители шаблона регулярного выражения, 388, 392
разделитель имен путей в UNIX, 509
:: (двоеточие), оператор доступа к методам или к членам-переменным класса, 181
; (точка с запятой)
разделитель аргументов URL, 212
символ конца оператора, 25
<> (угловые скобки)
< и >, преобразование в элементы HTML, 268, 336
<? и ?>, открывающие и закрывающие теги PHP, 527
соглашения по программированию в этой книге, 20
<<<, во встроенных документах, 24, 35
>&, оператор (перенаправление), 539
= (знак равенства)
=&, оператор присваивания, 169, 186

=, оператор присваивания,против оператора равенства = =, 146
= =, оператор равенства, 120
= = =, оператор тождества, 147
=>, оператор
инструктирование массивов использовать другой и, 104
определение пар ключ/значение для массивов, 103
#! (решетка-восклицание), синтаксис, PHP-сценарии, начинающиеся с, 572
? (знак вопроса)
метасимвол регулярного выражения, 388
?:, оператор (трехчленный), 147
после квантификаторов, для непоглощающего сравнения, 395
символ-заместитель в запросах к базам данных, 292
@ (знак at) в командах PHPDoc, 620
\ (обратная косая черта)
преобразование в escape-последовательности в SQL и групповые символы для замены имен в оболочке, 296
преобразование в escape-последовательности разделителей шаблона-регулярного выражения, 392
разделитель имен путей в Windows, 510
строка в двойных кавычках, escape-последовательность, 24
^ (знак вставки)
инвертирующий символьный класс, 389
привязка шаблона к началу строки, 389
_ (подчеркивание)
__FILE__ и __LINE__, константы, 227
групповой символ в SQL, 296
замена точки в именах переменных, 269
синоним _() для функции gettext(), 475
{} (фигурные скобки)
в интерполяции переменных, 484
выражения в, оценка в PHP, 149
пустое множество, 137
разадресация объектов, 365
строки в двойных кавычках, 24
элементы массива, разрешение неопределенности в, 150
| (вертикальная черта),
для определения альтернатив при сравнении с шаблоном, 390
~ (логическое НЕТ), 223
‘, оператор (обратный апостроф), 538
’ (одинарные кавычки)
в аргументах командной строки, 536
в значениях, возвращаемых функцией, 173
в строках, 23
преобразование в escape-последовательности для SQL, 278
->, оператор доступа к методам или к членам-переменным, 181

Числа

0 (нуль)
в строках с escape-последовательностями, 24
значения, возвращаемые функциями, 173
как пустая переменная, 145
числа, начинающиеся с, 87

А

Accept-Encoding, заголовок, 221
Accept-Language, заголовок, 473
access_log, файл, переменные в, 235
activate, сигнал, 593
add(), метод
GtkVBox, класс, 588
GtkWindow, класс, 586
addcslashes(), функция, 392
addHeader(), метод (HTTP_Request), 327, 329
addObjectMap(), функция (SOAP_Server), 381
addslashes(), функция, 155
ADOdb, уровень абстракции базы данных, 286
affectedRows(), метод (DB), 294
allow_call_time_pass_reference, конфигурационная директива, 165
allow_url_fopen, параметр настройки, 324
analog (программа статистики веб-сайта), 343

Apache, веб-серверы
взаимодействие внутри, 235
apache_note(), функция, 236
append(), функция
GtkMenu, класс, 592
GtkMenuBar, класс, 592
append_child(), метод, 356
array(), функция, 102
другой индекс, синтаксиса =>, 104
построение массивов из отдельных значений, 148
смешивание и сопоставление числовых и строковых ключей в, 105
array_diff(), функция, 135, 194
объединение с функцией array_map(), 136
array_filter(), функция, 122, 123
array_flip(), функция, 120
array_intersection(), функция, 135
array_keys(), функция, 134, 194
array_map(), функция, 109
объединение с функцией array_diff(), 136
array_merge(), функция, 114, 135, 137
array_multisort(), функция, 129
array_pad(), 112
array_pop(), функция, 102, 112
извлечение последнего элемента при объединении массивов, 118
array_push(), функция, 102, 137
соединение массивов и, 115
array_reverse(), функция, 124, 522
array_search(), функция, 121
array_shift(), функция, 112
array_splice(), функция, 111, 114
array_unique(), функция, 133, 134, 135
array_values(), функция, 111, 134
array_walk(), функция, 177
arsort(), функция, 124, 126
ASCII
значения символов в шаблонах или заместителях ereg, 393
коды символов, 475
параметр FTP_ASCII, 496
преобразование HTML в, 337
преобразование в шестнадцатеричную форму для функций preg, 393
преобразование в/из HTML, 336
символы-ограничители строки в Windows, 509
asin(), функция, 60
asort(), функция, 124, 125
asp_tags, параметр конфигурации, 235
atan2(), функция, 59
atanh(), функция, 60
attach(), метод (GtkTable), 589
Auth, класс, 499, 500, 501
auto_prepend_file, параметр конфигурации, 229
autoPrepare(), метод (DB), 302

B

\b (граница слова), метасимвол регулярного выражения, 334, 390
base_convert(), функция, 62, 552
base64_decode(), функция, 410
base64_encode(), функция, 410
basename(), функция, 264, 554
BCMath, библиотека арифметическая, 61
Benchmark, модуль, 237
Benchmark_Iterate, класс, 237
Benchmark_Iterate::get(), метод, 238
bindec(), функция, 63
bindtextdomain(), функция, 474
/bin/stty (управление характеристиками терминала), 584
break, оператор, 123
browscap, параметр конфигурации, 210
BSD, системы, локали в, 460
build(), метод (DB_Pager), 307

C

C, язык
PECL (PHP Extension Code Library), библиотека дополнительных программ PHP, 607, 616
call(), метод (SOAP_Client), 377
catalog-compare.php, программа, 472
CBC (Cipher Block Chaining), режим цепочки кодовых блоков, 421
checkdate(), функция, 83
chgrp(), функция, 554
child_nodes(), 359
chmod(), функция, 549, 553
chop(), функция, 36
chown(), функция, 554
chr(), функция (Windows), 584
class, ключевое слово, 179
clearstatcache(), функция, 553
close(), функция, 547

closedir(), функция, 547
combine(), функция, 191
command-shell.php, программа, 595, 598
compact(), функция, 371
config-get php_dir, команда (PEAR), 611
config-help, команда (PEAR), 612
connect(), метод
  DB, класс, 284
  GtkButton, класс, 590
  класс Cache_DB, 310
connect(), функция
  класс DB, 277
Console_Getopt, класс, 578, 581
  readPHPArgv(), метод, 581
__construct(), имя конструктора в Zend Engine 2, 184
Content-Length, заголовок, 325
Content-Type, заголовок изображения, передача броузеру, 453
Cookie, заголовок, 327
$_COOKIE, суперглобальный массив, 201, 266
cookies, 198
  аутентификация, основанная на, 216, 218
  добавление проверочного хеша в, 411
  запись определенного cookie в файл, 507
  информация о характеристиках броузера и, 210
  отслеживание сеанса с помощью, 204
  перенаправление по другому адресу, 203
  получение содержимого URL с помощью, 326, 328
  посылка только через соединения SSL, 429
  сообщение об ошибке /headersalreadysent/ (заголовки уже посланы), 227
  удаление, 202
    время истечения срока действия и другие значения, 202
  установка, 200
    SSL-соединения, определение, 201
    время истечения срока действия, 200
    домен, определение, 200
    пути к страницам, 200
  чтение значений, 201
copy(), функция, 556
cos(), функция, 59
cosh(), функция, 60
count(), повторное вычисление в циклах по массивам, 109
create_function(), функция, 177
create_text_node(), функция, 356
crypt(), функция, 407
  одностороннее шифрование, использование, 413
  шифрование паролей, 412
CSV (см. значения, разделенные запятой), 36
cURL расширение
  извлечение содержимого URL с помощью заголовков, 328
cURL, расширение
  включение заголовков ответа в вывод функции curl_exec(), 331
  возможность cookie jar, 327
  загрузка и использование с помощью FTP, 496
  извлечение содержимого URL, 322
  извлечение содержимого веб-страниц программой stale-links.php, 343
  имя пользователя и пароль в URL, 322
  перенаправление на другую веб-страницу, 323
curl_close(), функция, 497
curl_exec(), функция, 323, 497
curl_init(), функция, 496
curl_setopt(), функция, 496
CURLOPT_COOKIE, параметр, 327
CURLOPT_HEADER, параметр, 331
CURLOPT_HTTPHEADER, параметр, 328

D

\d, цифровой метасимвол, 390
database_interface (DSN), 285
date(), функция, 66, 67, 69, 271
  день недели, месяца или года, неделя года, 81

преобразование метки даты/времени UNIX в понятный человеку формат, 493
форматирование дат и времени, 72
форматирование даты и времени, 77
DAYOFWEEK(), функция (MySQL), 82
DB::affectedRows(), метод, 294
DB::autoPrepare(), метод, 302
DB::connect(), функция, 277
DB::execute(), функция, 291, 292
DB::getAll(), метод, 289
DB::getAssoc(), метод, 289
DB::getCol(), метод, 289
DB::getOne(), метод, 289
DB::getRow(), метод, 288
DB::isError(), метод, 297
DB::modifyLimitQuery(), метод, 308
DB::nextId(), функция, 300
DB::prepare(), функция, 291, 292
DB::query(), метод, 285, 298
DB::query(), функция, 291
DB::quote(), метод, 287, 295
вызов strtr() после, 296
DB::setFetchMode(), метод, 288
DB_Error, класс, 287, 297
DB_Pager, класс, 306
fetchRow(), метод, 307
DB_Pager::build(), метод, 307
DB_Result::fetchInto(), функция, 286
DB_Result::fetchRow(), функция, 286
DB_Result::numRows(), метод, 294
dba_close(), функция, 282
dba_exists(), функция, 282
dba_fetch(), функция, 282
dba_firstkey(), функция, 282
dba_nextkey(), функция, 282
dba_open(), функция, 282
dba_optimize(), функция, 283
dba_replace(), функция, 282
dba_sync(), функция, 283
decbin(), функция, 63
dechex(), функция, 63
declare, конструкция, 238
decoct(), функция, 63
\ddd (восьмеричные коды символов), 394
deg2rad(), функция, 60
DELETE, команда (SQL), 291
определение количества строк, возвращенных запросом, 294
delete-user.php, программа, 240, 242
destroy, сигнал, 587
/dev/null, псевдоустройство (UNIX), перенаправление вывода в, 540
dgettext(), функция, 474
dir, метод, 564
dir(), функция, 547, 557
dirname(), функция, 554
display(), функция, 237
dl(), функция, 576
загрузка пакетов PECL, 617
DNS (Domain Name Service), служба доменных имен
Net_DNS, пакет, 503
выполнение поиска, 502
запросы динамического обновления, 537
проверка действительности адресов электронной почты, 400
DOM
libxml, анализатор, 355
анализ XML с помощью, 357, 360
анализ на основе дерева, 358
поиск определенных элементов в дереве, 359
генерация XML с помощью, 354, 357
спецификация W3C, веб-сайт для, 359
узел верхнего уровня или корневой узел, 356
DSN (имя источника данных), 206, 285
DST (см. летнее время, переход на), 93
dump_file(), метод
документы DOM XML, 357
dump_file(), функция
генерация корректного XML-документа, 354
dump_mem(), метод
форматирование вывода DOM XML, 357
dump_mem(), функция
генерация корректного XML-документа, 354

E

e, число, возведение в степень, 56
E_USER_ERROR, ошибка, 299
E_USER_NOTICE, внутренняя ошибка, 299
E_USER_WARNING, ошибка, 299

each(), функция
редко заполненные массивы, 135
ECB (Electronic Code Book), режим электронной кодовой книги, 421
echo, 533
enable-ftp, параметр, 495
end_element(), метод, 365
$_ENV, суперглобальный массив, 231
ereg(), функция, 87
ereg, функции
функции preg и, 388
ereg_replace(), функция, 393
eregi(), функция
сравнение, не чувствительное к регистру, 392
error_log(), функция, 226
error_reporting(), функция, 223
escapeshellarg(), функция, 535
eval(), функция, 195
Event Log (Windows NT), 222
execute(), метод, 291
execute(), функция (DB), 291, 292
executeMultiple(), функция (DB), 293
exp(), функция, 56
explode(), функция
выделение строк вместо регулярных выражений, 406
выделение хеша из значения cookie, 411
генерирование частей времени с помощью функции date(), 77
графика гистограммы, размерные линии, 457
данные, разделенные запятыми и, 37
строки, разбиение на части, 40
ext/standard/parsedate.y, 85
extends, ключевое слово, 181

F

false, значения, 173
fclose(), функция, 489, 508
feof(), функция, 518
fetchInto(), метод
DB, класс, 287, 310
fetchInto(), функция
DB_Result, класс, 286
fetchRow(), метод
DB, класс, 285, 310
DB_Pager, класс, 307
fetchRow(), функция
DB_Result, класс, 286
fflush(), функция, 533
fgetcsv(), функция, 36
пропуск определенных возвращаемых значений, 172
fgets(), функция, 510
аргумент длины строки для записей переменной длины, 525
подсчет строк, абзацев или записей в файле, 517
fgetss(), функция, 338
__FILE__, константа, 227
file(), функция, 164, 401, 511, 522
чтение файла в массив, 524, 529
file_exists(), функция, 543
fileatime(), функция, 550
filectime(), функция, 550
filemtime(), функция, 516, 550
$_FILES, суперглобальный массив
обработка загруженного файла, 263
filesize(), функция, 515
flock(), функция, 280
консультативная блокировка с помощью, 540
отмена блокировок, 541
flush(), метод
Cache_DB, класс, 311
flush(), функция, 218
fopen(), функция, 36, 320, 322, 507
stdin, поток, открытие, 515
временные файлы, 513
дескриптор файла для записи заголовков ответа в файл, 332
значение прав доступа по умолчанию, 550
имя пользователя и пароль в URL, 322
перенаправления, следование, 323
режимы файлов, 512
стандартные потоки I/O, использование с помощью, 574, 582
удаленные файлы, 514
for, циклы
вместо функции range(), 51
выполнение итерации по массиву информации о файле, 553
выполнения цикла по вхождениям адресов LDAP, 498
изменение порядка элементов массива на обратный, 124

итерация по массивам, 107
шестнадцатеричные числа в, 64
foreach, циклы, 51
выполнение цикла по массиву и выполнение операций с элементами, 103
итерации по массивам, 107
нахождение элементов массива, удовлетворяющих определенным требованиям, 122
определение всех комбинаций элементов массива, 137
редко заполненные массивы, 135
соединение массивов, 117
выполнение цикла по массиву и выполнение операций с элементами, 103
formatter-pdf, программа, 536
fpassthru(), функция, 516
fputs(), функция, 537
fread(), функция, 322, 510, 515
FreeType, библиотека, 442, 448
frenchtojd(), функция, 97
fresh-links.php, программа, 345, 348
fseek(), функция, 528, 529
fsockopen(), функция, 320, 325, 488
fstat(), функция, 553
ftell(), функция, 528
FTP, протокол, 477
cURL, расширение, использование с помощью, 496
анонимный FTP, 495
загрузка и выгрузка файла по существующему указателю открытого файла, 496
копирование файлов между удаленным сервером и вашим компьютером, 495
перемещение файлов с помощью, 494, 497
ftp_close(), функция, 496
ftp_fget(), функция, 496
ftp_fput(), функция, 496
ftp_get(), функция, 495
ftp_put(), функция, 495
ftp_set_option(), функция, 496
ftruncate(), функция, 531
func_get_arg(), функция, 168
func_num_args(), функция, 168
function, ключевое слово, 179
function_defined(), 62
fwrite(), функция, 508, 535, 537

G

GD, библиотека
прозрачный цвет, 452
gdbm_reorganize(), функция, 284
GET, метод
получение содержимого URL с помощью, 324
получение удаленного файла, 514
$_GET, суперглобальный массив, 266
get_browser(), функция, 210
get_cfg_var(), функция, 234
get_class_methods(), функция, 193
get_class_vars(), функция, 193
get_elements_by_tagname(), 359
get_gmtime(), 374
get_html_translation_table(), функция, 268
get_lock(), метод, 245
get_object_vars(), функция, 193
get_parent_class(), функция, 188
get_time(), метод, 374
getAll(), метод (DB), 289
getAssoc(), метод (DB), 289
getChannelInfo() (XML_RSS), 385
_getch(), функция, 584
getCol(), метод (DB), 289
getdate(), функция, 67, 69, 71
getenv(), функция, 231
gethostbyaddr(), функция, 502
gethostbyname(), функция, 502
gethostbynamel(), функция, 502
getItems(), метод (XML_RSS), 385
getMessage(), метод (DB_Error), 297
getmicrotime(), функция, 231
getmxrr(), функция, 503
getOne(), метод (DB), 289
getopt(), метод (Console_Getopt), 578, 581
Getopt_Error, класс, 581
getProxy(), метод (SOAP_WSDL), 379
getResponseBody(), метод (HTTP_Request), 331
getResponseHeader(), метод (HTTP_Request), 331
getResult(), метод (DB), 181
getRow(), метод (DB), 288
gettext, расширение, 474
GlobalWeather, веб-служба, 598

gmdate(), функция, 76, 89
gmmktime(), функция, 70, 71
GMP, библиотека, 61
gmstrftime(), функция, 76, 89
GNU
  gettext, утилиты, 474
  Privacy Guard (GPG), 429
  Readline, библиотека, 583
Google
  объекты SOAP, 378
  объекты SOAP_WSDL, 379
go-pear, сценарий, 609
GPG (GNU Privacy Guard), криптозащита, 429
gregoriantojd(), функция, 79, 97
grid_horizontal(), функция, 142, 144
gtk, 586, 587
GTK+, онлайновая документация по, 576
GtkButton, класс, 586
GtkMenu, класс, 592
GtkMenuBar, класс, 592
GtkMenuItem, класс, 592
GtkTable, класс, 589
  attach(), метод, 589
GtkVBox, класс, 588
gtk-weather.php, программа, 598, 606
GtkWindow, класс, 586
GUI (Graphical User Interface), графический интерфейс пользователя
  реакция на действия пользователей, 590
  элементы управления окном
    показ в окне, 586
    создание и показ, 575
gunzip, утилита, 544
gzcompress(), функция, 544
gzencode(), функция, 544
gzgets(), функция, 544
gzopen(), функция, 543
gzread(), функция, 543
gzseek(), функция, 544
gzuncompress(), функция, 544
gzwrite(), функция, 543

H

HEAD, метод, 343, 346
header(), функция, 203, 353, 452
  изображение PNG, передача броузеру, 435
  устранение ошибок /headersalreadysent/ (заголовки уже посланы), 227
heredocs, встроенные документы, инициализация строк с помощью, 23
  строки в, 24
hexdec(), функция, 63
HOME, переменная окружения, 430
HTML, язык разметки
  XML и, 349
  анализ, применение непоглощающего сравнения, 397
  вывод на печать массива в виде горизонтально расположенных столбцов, 141
  выделение текста, заключенного в теги, 402
  выделенный текст, преобразование с помощью тегов <b> или <i>, 531
  значения цветов, вывод на печать, 63
  извлечение ссылок из файла, 334
  поиск тега изображения с помощью регулярных выражений, 388
  преобразование в ASCII, 336
  преобразование специальных символов в сообщениях новостных групп в escape-последовательности, 494
  данные о регистрациях в UNIX, в виде таблицы, 538
  теги, удаление из строки или из файла, 338
  шаблоны Smarty для, 339
  электронная почта, содержащая, 481
  элементы, 336
html_errors, переменная конфигурации, 235
htmlentities(), функция, 212, 267, 336
htmlspecialchars(), функция, 268, 364, 494
HTTP, протокол
  базовая аутентификация, 213, 215
  заголовки (см. заголовки, HTTP), 203
  запросы, отсутствие памяти, 198
  коды статуса
    Request-URI Too Long (запрос URI слишком длинный) (414), 326

веб-страница была перемещена (код 302), 346
программа не может извлечь содержимое URL, 345
методы, 322, 324, 326
GET, 203, 330, 514
HEAD, 343, 346
POST, 203, 331, 373, 381, 500
обмен информацией с использованием SOAP поверх, 377
отладка цикла запрос/ответ, 330
прокси-сервер, конфигурация PEAR для использования с, 611
смешивание заголовков и текста тела в буферизованном выводе, 228
чтение посредством, 514
HTTP_Request, класс, 321
addHeader(), метод, 327, 329
getResponseHeader() и getResponseBody(), методы, 331
извлечение содержимого URL, 322, 346
имя пользователя и пароль в URL, 323
перенаправление и, 323
HTTP_USER_AGENT, переменная окружения, 210
httpd.conf, файл, 232
идентификатор сеанса, запись в access_log каждого запроса, 236
HTTPS (HyperText Transmission Protocol, Secure) протокол защищенной передачи гипертекстов, 329

I

I18N (см. интернационализация), 458
ibase_execute(), функция, 294
ibase_prepare(), функция, 294
ImageArc(), функция, 438
ImageColorAllocate(), функция, 435
ImageColorsForIndex(), функция, 452
ImageColorTransparent(), функция, 451
ImageCreate(), функция, 434
ImageCreateFrom(), функция, 192
ImageCreateFromPNG(), функция, 434
ImageDestroy(), функция, 436, 457
ImageEllipse(), функция, 439
ImageFilledArc(), функция, 439
ImageFilledEllipse(), функция, 439
ImageFilledPolygon(), функция, 437
ImageFilledRectangle(), функция, 435, 436
отрисовка гистограммы, 457
ImageFillToBorder(), функция, 439
ImageFtBBox(), функция, 448
ImageLine(), функция
координаты линий, 437
ImagePNG(), функция, 457
ImagePolygon(), функция, 436
координаты многоугольников, 437
ImagePSBBox(), функция, 446
ImagePSFreeFont(), функция, 443
ImagePSLoadFont(), функция, 441, 443
ImagePSText(), функция, 441, 443
ImageRectangle(), функция, 436
координаты прямоугольников, 437
ImageSetStyle(), функция, 440
ImageString(), функция, 441
координаты, 448
текст для гистограммы, 457
ImageStringUp(), функция, 442
ImageSX(), функция, 446
ImageSY(), функция, 446
ImageTTFBBox(), функция, 448
ImageTTFText(), функция, 441
параметры, порядок, 442
imagetypes(), функция, 434
IMAP, расширение, 483
NNTP, протокол, 489
значения MIME-типа, 486
imap_body(), функция, 485
imap_close(), функция, 494
imap_fetchbody(), функция, 485, 494
imap_fetchstructure(), функция, 485
imap_header(), функция, 194
поля NNTP-сервера, 491
imap_headers(), функция, 485
imap_mail_compose(), функция, 487, 488
imap_num_msg(), функция, 490
imap_open(), функция, 484
соединения с NNTP-сервером, 490
imap_rfc822_parse_adrlist(), функция, 398
<img>, тег (локализованный), 471
img(), функция-оболочка, 470
in_array(), функция, 119, 121
двойные элементы и, 134
include, параметр, 324
include_path, директива, 472

проверка для пакетов PEAR, 611
info, команда (PEAR), 613
ini_get(), функция, 233, 611
ini_get_all(), функция, 233
ini_set(), функция, 235
INSERT, команда (SQL), 291
запрос уникального идентификатора базы данных, 301
определение количества строк, возвращенных запросом, 294
install, команда(PEAR), 617
IP-адреса, поиск с помощью DNS, 502
is_array(), функция, 109
is_bool(), функция, 49
is_double(), функция, 49
is_file(), функция, 557
is_float(), функция, 49
is_int(), функция, 49
is_long(), функция, 49
is_numeric(), функция, 48, 49
is_readable(), функция, 516
is_real(), функция, 49
is_uploaded_file(), функция, 264
isError(), метод (DB), 297
isset(), функция, 119, 145, 166
присваивание переменным значений по умолчанию, 147
i-узлы (inodes), 547
функции, предоставляющие информацию о файле, 548

J

jdtofrench(), функция, 97
jdtogregorian(), функция, 97
jdtojewish(), функция, 98
jdtojulian(), функция, 97
jdtounix(), функция, 88
jewishtojd(), функция, 98
join(), функция
добавление проверочного хеша в cookie, 411
пустые строки, используемые с, 164
соединение массивов, 116
создание строки запроса GET, 211
JPEG, 432
juliantojd(), функция, 97

L

L10N (см. локализация), 458
last, команда (UNIX), 538
LC_ALL, переменная окружения, 462
LDAP (Lightweight Directory Access Protocol), облегченный протокол доступа к справочникам, 497
аутентификация пользователей с помощью, 499, 501
поиск адресов, 497
сервер
взаимодействие с, 498
загрузка, 498
хранилище адресов (источник данных), 498
ldap_bind(), функция, 498
ldap_get_entries(), функция, 498
ldap_list(), функция, 499
ldap_search(), функция, 498
libxml для DOM-анализа, 355
libxml для анализа DOM, 359
__LINE__, константа, 227
Linux
locale, программа, 460
оператор ` (обратный апостроф), использование в, 538
list(), функция
присваивание значений из массива отдельным переменным, 148
пропуск определенных значений, возвращаемых функцией, 172
разделение массива на отдельные переменные, 104, 170
list, команда (PEAR), 610
list-upgrades, команда (pear), 618
locale, программа, 460
localeconv(), функция, 467
информация, связанная с денежным обращением, 468
localtime(), функция, 67, 68, 69
текущий статус поддержки DST, 93
Location, заголовок, 323
log(), функция, 55
log_errors, параметр конфигурации, 221
log10(), функция, 55
ls, команда (UNIX)
восьмеричное представление прав доступа к файлу, 552
lstat(), функция, 553
ltrim(), функция, 35

M

Mail, класс, 478, 479
Mail_mime, класс, 481, 482, 483
mailto:гиперссылки, преобразование текстовых адресов электронной почты в, 387
man strftime, 77
mangle_email(), функция, 220
max(), функция, 123
mcrypt, расширение, 408, 417
  версии 2.2 и 2.4, 422
  перечень алгоритмов шифрования/дешифрования, 418
mcrypt_create_iv(), функция, 422
mcrypt_decrypt(), функция, 418
mcrypt_encrypt(), функция, 418
mcrypt_get_block_size(), функция, 422
mcrypt_get_iv_size(), функция, 422, 426
mcrypt_list_algorithms(), функция, 418
mcrypt_list_modes(), функция, 418
md5(), функция, 407
mean(), функция, 168
  передача массива с переменными аргументами, 167, 169
message.php, программа, 313, 319
Message-ID, идентификатор сообщения, 489
Metabase, уровень абстракции базы данных, 286
method_exists(), функция, 302
mhash, модуль, 412
Microsoft
  msvcrt.dll, C-библиотека времени выполнения, 584
  Windows (см. системы Windows), 516
microtime(), функция, 53, 94, 231
MIME-типы
  в сообщениях новостных групп, 488
  передача броузеру типа файла изображения, 517
min(), функция, 123
mkdir(), функция, 542, 543, 550, 561
mktime(), функция, 71, 271
  даты до начала эпохи, обработка, 78
  преобразование дат и времени часового пояса в метку времени UNIX, 70
mode_string(), функция, 564
modifyLimitQuery(), метод (DB), 308
move_uploaded_file(), функция, 264
msg(), функция, 463
  объединение с функцией sprintf(), функция, 463
msvcrt.dll, C-библиотека времени выполнения, 584
mt_getrandmax(), функция, 52
mt_rand(), функция, 52
MX-записи, получение, 503
MySQL, 276
  DAYOFWEEK(), функция, 82
  UNIX_TIMESTAMP(), функция, 79
  WEEK(), функция, 82
  WEEKDAY(), функция, 82
  доска сообщений, разделенных на потоки, 319
  команда REPLACE INTO, 208
  наследование в, 181
mysql_connect(), функция, 276
mysql_fetch_object(), функция, 194
mysql_fetch_row(), функция, 276
mysql_pconnect(), функция, 277
mysql_query(), функция, 276
mysql_select_db(), функция, 276

N

natsort(), функция, 126
Net_DNS, класс, 503
Net_Ping, класс, 503, 504
Net_Whois, класс, 505
new, ключевое слово, 179
  реализация объектов, 182
news.php.net, сервер для почтовых списков PHP, 490
nextId(), функция (DB), 300
nl2br(), функция, 494
NNTP (Network News Transfer Protocol) сетевой протокол передачи новостей, 489, 490, 494
  imap_header(), функция поля NNTP-сервера, 491
notify-user.php, программа, 240, 241
NOW(), функция (SQL), 293
nsupdate, команда, 537
NUL, псевдоустройство
  в Windows, перенаправление вывода в, 540
number_format(), функция, 57, 467

numRows(), метод (DB_Result), 294
n-е совпадение, нахождение, 394

O

ob_end_flush(), функция, 219
ob_start(), функция, 219
OCIBindByName(), функция, 294
OCIExecute(), функция, 275, 294
OCIFetch(), функция, 275
OCILogin(), функция, 275
OCIParse(), функция, 275, 294
OCIPLogon(), функция, 277
OCIResult(), функция, 275
octdec(), функция, 63
opendir(), функция, 547, 557
OpenSSL, библиотека, 330
Oracle, базы данных
  OCI8 backend, 285
  извлечение данных из, 275
ORACLE_SID, переменная окружения, 275
output_buffering, параметр конфигурации, 220
output_handler, параметр конфигурации, 220

P

pack(), функция
  символы форматирования, перечень, 45
  сохранение двоичных данных в строках, 44
  строки формата, 44
pack_start(), метод (GtkVBox), 588
package, команда (PEAR), 611
$parent(), функция, 188
parent::, префикс имени метода, 182
parent_node(), метод, 359
parse_ini_file(), функция, 525
  по разделам, 526
pathinfo(), функция, 554
pc_array_power_set(), функция, 136
pc_array_shuffle(), функция, 131, 132, 524
pc_array_to_comma_string(), функция, 106, 118
pc_ascii2html(), функция, 336
pc_assign_defaults(), функция, 166
pc_auth_ldap_signin(), функция, 500
pc_bar_chart(), функция, 454
pc_build_query(), функция, 302
pc_calendar(), функция, 98, 100
pc_check_the_count(), функция, 151
pc_checkbirthdate(), функция, 83
pc_date_sort(), функция, 127
pc_DB_Session, класс, 205, 206, 209
pc_DB_Session::_write(), метод, 208
pc_debug(), функция, 229
pc_decode(), функция, 257
pc_encode(), кодирующая функция, 257
pc_error_handler(), функция, 226
pc_fixed_width_substr(), функция, 38
pc_format_currency(), функция, 467, 469
pc_html2ascii(), функция, 337
pc_ImagePSCenter(), функция, 445, 446, 448
pc_ImageStringCenter(), функция, 444, 448
pc_ImageTTFCenter(), функция, 445, 448
pc_indexed_links(), функция, 306, 307
pc_link_extractor(), функция, 334, 343, 346
pc_log_db_error(), функция, 164
pc_logn(), функция, 55
pc_mail(), функция, 480
pc_may_pluralize(), функция, 58
pc_MC_Base, класс, 465
pc_message_save(), функция, 318
pc_mkdir_parents(), функция, 545, 562
pc_mktime(), функция, 90
pc_multi_fwrite(), функция, 534
pc_next_permutation(), функция, 139
pc_passwordcheck(), функция, 414, 416
pc_permute(), функция, 139
pc_post_request(), функция, 325
pc_print_address(), функция, 195
pc_print_link(), функция, 306, 307
pc_process_dir(), функция, 559
pc_process_dir2(), функция, 560
pc_randomint(), функция, 523
pc_RSS_item::character_data(), метод, 363, 365
pc_RSS_item::display(), метод, 364
pc_RSS_item::end_element(), метод, 363
pc_RSS_item::start_element(), метод, 363
pc_RSS_parser, класс, 362

pc_split_paragraphs(), функция, 518
pc_split_paragraphs_largefile(), функция, 518
pc_tab_unexpand, функция, 31
$pc_timezones, массив, 89
pc_user, класс, 189
pc_validate(), функция, 213, 216
pc_validate_zipcode(), функция, 253
pc_Web_Abuse_Check, класс, 243, 245
pclose(), функция, 537, 538
PEAR (PHP Extension and Add-on Repository), 607–622
  Auth, класс, 499
  Benchmark, модуль, 237
  Cache_DB, пакет, 310
  Console_Getopt, класс, 578, 581
  DB_Pager, класс, 306
  HTTP_Request, класс, 321
  Mail, класс, 478
  Mail_mime, класс, 481
  Net_DNS, пакет, 503
  Net_Ping, пакет, 503
  Net_Whois, класс, 505
  SOAP, классы, 376
  SOAP_Server, класс, 380
  XML_RSS, класс, 384
  XML_Transform, пакет, 363
  более ранние версии, проблемы, 608
  документирование классов с помощью PHPDoc, 620
  инсталляция в UNIX и Windows, 609
  инсталляция пакетов PECL, 616
  информация на веб-сайте о, 607
  команды, перечень, 610
  конфигурация для использования с прокси-сервером HTTP, 611
  менеджер пакетов (см. pear, приложение), 610
  нахождение пакетов, 612
  обновление пакетов, 618
  пакеты, определение местоположения, 611
  руководство, веб-сайт для, 609
  сбор информации о пакете, 613
  удаление пакетов, 619
  уровень абстракции базы данных DB, 277
    (см. также DB), 284
    машины баз данных, поддерживаемые, 285
  установка пакетов, 615
pear, приложение, 607, 610
  info, команда, 613
  install, команда, 617
  list-upgrades, команда, 618
  phpize, команда, 617
  remote-info, команда, 614
  remote-list, команда, 612
  search, команда, 612
  uninstall, команда, 619
  upgrade, команда, 619
  более ранние версии, проблемы с, 608
  загрузка и инсталляция пакетов с сервера, 615
  инсталляция, 609
  команды, 611
  настройки конфигурации, 611
PEAR::Error, класс, 297
PEAR_Error, базовый класс, 581
PEAR_ERROR_CALLBACK, константа, 298
PEAR_ERROR_DIE, константа, 298
PEAR_ERROR_PRINT, константа, 298
PECL (PHP Extension Code Library), библиотека дополнительных программ PHP, 607
  инсталляция пакетов, 616
Perl, язык
  chop() (не рекомендуется), 36
  регулярные выражения, совместимые с
    веб-сайт для, 391
    функции preg для, 388
pg_connect(), функция, 276
pg_exec(), функция, 276
pg_fetch_row(), функция, 276
pg_numrows(), функция, 276
pg_pconnect(), функция, 277
PGP (Pretty Good Privacy), программа, 430
Phorum, пакет доски сообщений, 319
  Zend Engine 2 (ZE2), 182
  веб-сайты справочных сведений, 18
  генераторы случайных чисел, 53
  материалы по DOM XML в онлайновом руководстве по PHP, 358
  пакеты доски сообщений, 319
  поддержка LDAP, 498
  поддержка Unicode, 460
  расширение DOM XML, 354

реляционные базы данных, поддерживаемые, 273
PHP на стороне клиента, 572, 606
command-shell.php, программа, 595, 598
gtk-weather.php, программа, 598, 606
GUI-элементы управления окном, показ в окне, 586
анализ аргументов программы, 577
с помощью getopt(), 578, 581
показ меню в окн, 592, 595
показ нескольких GUI-элементов управления в окне, 587, 590
реакция на действия пользователя, 590
чтение паролей без визуализации, 583, 586
чтение с клавиатуры, 582
PHP, конфигурационные переменные
значения access, 234
установка, 235
php.ini, файл конфигурации, 223
browscap, параметр конфигурации, 210
include path для пакетов PEAR, 611
session.save_handler, установка для хранения сеанса пользователя, 205
конфигурационные значения для PHP, постоянные изменения в, 235
настройки почты, 479
проверка исходного значения переменной конфигурации, 234
чтение, 525
PHP_AUTH_PW, глобальная переменная, 213
PHP_AUTH_USER, глобальная переменная, 213
php_session, таблица, 206
php-cli, сценарий, 609
PHPDoc, 620
теги, 620
phpdoc, программа, 622
PHP-GTK, расширение, 576, 586
онлайновая документация, 576
показ панели меню в окне, 592, 595
phpinfo(), функция, 175
пароли, хранящиеся в переменных окружения, 409
проверка версии GD, 433
phpize, команда (pear), 617
PHP-программы командной строки, 572
CLI, интерпретатор командной строки
расширение PHP-GTK, запуск программ, которые используют, 575
$COMMAND_LINE, глобальная переменная, 573
php-cli, сценарий, 609
двоичный код интерфейса командной строки (CLI)
расширение PHP-GTK, запуск программ, которые используют, 575
ping, программа, 503
PNG, формат файла
изображение гистограммы, 457
POP3, протокол, 483
popen(), функция, 537, 538, 539
POSIX
регулярные выражения, информация веб-сайта о, 391
функции регулярных выражений, 388
POST, метод
максимальный размер файлов, 265
$_POST, суперглобальный массив, 249
post_max_size, параметр конфигурации, 265
PostgreSQL, база данных, 275
PostScript Type 1, шрифты, 441
t1lib, библиотека, 443
рисование центрированного текста, 445, 446
pow(), функция, 56
preg_grep(), функция, 401
preg_match(), функция, 388
анализ строк в комбинированном формате протокола, 341
анализ форматов даты, 87
имена файлов, соответствующих шаблону, 558
preg_match_all(), функция, 389
вывод программы ping, анализ, 504
занесение всех совпадений в массив, 394
preg_quote(), функция, 404
preg_replace(), функция, 389

preg_split(), функция, 40, 405
prepare(), функция (DB), 291, 292
print(), функция, 533
print_r(), функция, 156, 159
    вывод на печать значений переменных объекта, 194
    показ всего содержимого объекта Error, 297
printf(), функция, 564
    шестнадцатеричные числа, форматирование, 64
profile(), функция, 239
putenv(), функция, 232
    вызов перед вызовом функции mktime(), 90

Q

qmail, программа, 479
query(), метод
    Cache_DB, класс, 311
    DB, класс, 285, 291, 298
quote(), метод (DB), 287, 295
    вызов strtr() после, 296
quotemeta(), функция, 404

R

r, режим
    открытие канала в, 539
\r (возврат каретки), 393
    строки в двойных кавычках, 24
rad2deg(), функция, 60
rand(), функция, 53
range(), функция, 51, 107
RDF Site Summary (см. RSS-рассылки), 384
read(), функция, 547
readdir(), функция, 547, 557
readfile(), функция, 516
readline(), функция, 582
Readline, расширение, 582
readline_add_history(), функция, 583
readPHPArgv(), метод (Console_Getopt), 581
References, заголовок, 489
Referer, заголовок, 329
register_globals, параметр конфигурации, 235, 250
    запрещение в целях безопасности, 266
register_tick_function(), функция, 239
$_REQUEST, суперглобальный массив, 266
remote-info, команда (pear), 614
remote-list, команда (pear), 612
rename(), функция, 556
require, параметр, 324
reset(), функция
    возвращение значения первого элемента массива, 118
    перемещение указателя обратно в начало массива, 108
return_time(), 373
    связь методов XML-RPC с, 374
rewind(), функция, 528, 529
rm -rf, команда, 536
rmdir(), функция, 563
round(), функция, 50
RPC (см. XML, XML-RPC), 369
rsort(), функция, 126
RSS-рассылки
    обработка с использованием библиотеки expat и преобразование в HTML, 362
    чтение, 384, 386
rtrim(), функция, 35
    удаление символов новой строки с помощью, 525
    удаление символов-ограничителей строк, 509

S

\s (пробельный символ) метасимвол, 520
save-crypt.php, программа, 424
SAX, 360, 366
search, команда (pear), 612
SELECT, команда (SQL)
    кэширование результатов запроса, 311
    определение количества строк, возвращенных запросом, 294
sem_acquire(), функция, 153
sem_get(), функция, 152
sem_release(), функция, 153
sendmail, программа, 479
sendRequest(), функция (HTTP_Request), 346
serialize(), функция, 154, 257
    хранение сложных данных в БД, состоящей из текстовых файлов, 279

хранение сложных данных в файлах DBM, 283
service(), функция (SOAP_Server), 381
$_SESSION, суперглобальный массив, 204
session.entropy_file, параметр конфигурации, 218
session.entropy_length, параметр конфигурации, 218
session_name(), функция, 205
session_save_path(), функция, 205
session_set_save_handler(), функция, 206
session_start(), функция, 204, 256
set_attribute(), метод, 356
set_error_handler(), функция, 224, 225
set_submenu(), функция (GtkMenuItem), 592
set_text(), функция (GtkButton), 592
setcookie(), функция, 200
посылка только через соединения SSL, 429
удаление cookies, 202
устранение ошибок /headersalreadysent/ (заголовки уже посланы), 227
setErrorHandling(), функция (DB), 298
setFetchMode(), метод (DB), 288
setlocale(), функция, 460, 462
setMarker(), функция (Benchmark::Timer), 237
settype(), функция, 109
shm_attach(), функция, 153
shm_detach(), функция, 153
shm_get_var(), функция, 153
shm_put_var(), функция, 153
short_open_tag, параметр настройки, 353
show_all(), метод (GtkWindow), 586
shuffle(), функция, 131, 524
shutdown(), функция (GtkWindow), 586
Simple Object Access Protocol (см. SOAP), 377
sin(), функция, 59
sinh(), функция, 60
site-search.php, программа, 568, 571
SMTP-сервер, 479
SOAP (Simple Object Access Protocol), простой протокол доступа к объектам, 351
запросы, посылка, 376, 379
создание сервера и ответ на SOAP-запросы, 379, 382
SOAP_Server::addObjectMap(), функция, 381
SOAP_Server::service(), функция, 381
SOAP_WSDL::getProxy(), метод, 379
Solaris, системы, программа locale, 460
sort(), функция, 125
split(), функция, 40
sprintf(), функция
объединение с функцией msg(), 463
преобразование десятичных чисел в двоичные, восьмеричные и шестнадцатеричные, 63
форматирование значений результатов голосования, 457
SQL (Structured Query Language), язык структурированных запросов
базы данных
в паре с PHP, 274
выполнение запросов, 286
модификация данных в, 291
соединение с, 284
групповые символы, заключение в кавычки в запросах к базам данных, 296
переносимость, 277
специальные символы в, 278
srand(), функция, 422
SSL (Secure Sockets Layer), протокол защищенных сокетов, 428
cookies, посылка посредством, 201
cURL, расширение, использование с помощью, 329
соединения
IMAP и POP3, 484
stale-links.php, программа, 343, 345
start_element(), метод, 364
stat(), функция, 548, 551
вызов для символических ссылок, 553
информация о файле, возвращенная, 551
stdclass, класс, 178
добавление свойств к базовому объекту, 194
str_replace(), функция, 49
замена табуляций пробелами, и наоборот, 31
strace(), функция, 539

strftime(), функция, 67, 69
    день недели, месяца или года, неделя года, 81
    кнопка, показывающая текущее время, 591
    начальная дата для интервалов времени, 95
    строка формата %c (для локализации), 466
    форматирование даты и времени, 72, 77
strip_tags(), функция, 338, 397
stripslashes(), функция, 155
strnatcmp(), функция, 127
strpos(), функция, 173
strrev(), функция, 30
strtolower(), функция, 33
strtotime(), функция, 85
    анализ форматов даты, 87
    добавление или вычитание из даты, 87
strtoupper(), функция, 33
strtr(), функция, 296
substr(), функция, 26
    unpack(), функция, замена при извлечении полей фиксированной ширины, 39
    анализ записей фиксированной ширины в строках, 37
substr_replace(), функция, 27
switch, оператор, 151
    в эмуляции полиморфизма методов, 191
syslog(3), вызов (UNIX), 222
system(), функция, 536

T

t1lib, библиотека (шрифты PostScript Type 1), 443
tan(), функция, 59
tanh(), функция, 60
tempnam(), функция, 513, 555
textdomain(), функция, 474
thread_pos, поле, вычисление значения, 313
ticks, управляющий элемент (параметр), 239
ticks, элемент управления, 238
time(), функция, 95
time_parts(), функция, 170
tmpfile(), функция, 513
touch(), функция, 543
    время модификации файла, изменение, 550
    значение прав доступа по умолчанию для файлов, 550
    обновление времени модификации файлов, 551
track_vars, параметр конфигурации, 250
trigger_error(), функция, 299
trim(), функция, 35, 228
    регулярное выражение, эквивалентное, 390
TrueType, шрифты, 441
    FreeType, библиотека для, 442
    отрисовка центрированного текста, 445, 448

U

U, модификатор шаблона, 396
    квантификаторы, превращение в/из поглощающие и непоглщающие, 397
ucfirst(), функция, 32
ucwords(), функция, 32
umask(), функция, 550
/Undefinedvariable/, собщение об ошибке, 223
Unicode, 460
    чтение или запись, 475
uninstall, команда (pear), 619
uniqid(), функция, 261
UNIX
    man strftime, 77
    MySQL UNIX_TIMESTAMP(), функция, 79
    PEAR, инсталляция в, 609
    zoneinfo, библиотека, 90
        часовые пояса, перечисленные, 91
    загрузка и инсталляция PHP-GTK, 575
    имена путей в, 509
    перенаправление стандартного потока ошибок в стандартный поток вывода, 539
    права доступа к файлам, 549
    регистрации в системе, отформатированные в виде HTML-таблицы, 538

сравнение с именами файлов, использование *ls* с `, 558
файлы, ограничители строк и пути, 509
характеристики терминала, 584
unixtojd(), функция, 88
unlink(), функция, 556, 563
unpack(), функция
анализ записей фиксированной ширины в строках, 38
замена на функцию substr() при извлечении полей фикисированной ширины, 39
извлечение двоичных данных из строк, 44
символы форматирования, перечень, 45
строки формата, 44
unregister_tick_function(), функция, 240
unserialize(), функция, 154
unset(), функция, 145, 184, 267
переменные, помещенные в локальную область видимости с помощью ключевого слова global, 176
элементы массива, использование с, 110
UPDATE, команда (SQL), 291
определение количества строк, возвращенных запросом, 294
уникальный идентификатор базы данных, выполнение запросов, 301
update_time(), функция, 591, 595
upgrade, команда (pear), 619
URL
FTP, протокол, 496
внедрение идентификационной информации в, 514
защищенные паролем, 427
извлечение содержимого удаленных, 320, 330
cookies, получение с помощью, 326, 328
GET, метод, получение с помощью, 322–324
HTTPS, 329
POST, метод, получение с помощью, 324, 326
заголовки, получение с помощью, 328
кодирование, 211
urlencode(), функция, 154, 211
USER, переменная окружения, 430
User-Agent, заголовок, 329
usort(), функция, 126, 128, 177
/usr/bin/last, команда, 538
UTC (Universal Coordinated Time) всеобщее скоординированное время, 65
gmdate() и gmstrftime(), функции, 76
преобразование в метку времени UNIX, 70
смещения между часовыми поясами, 89
utf8_decode(), функция, 475
utf8_encode(), функция, 475

V

var, ключевое слово, 179
var_dump(), функция, 156, 159
просмотр переменных объекта, 194
рекурсия, обработка, 157
var_export(), функция, 194
variables_order, параметр конфигурации, 231
verify-user.php, страница, 240, 242

W

WBMP (Wireless BMP), формат , 432
WDDX (Web Distributed Data eXchange), технология обмена данными, 382, 384
wddx_serialize_value(), функция, 383
WSDL (Web Services Definition Language) язык определения веб-сервисов, 379
web-ls.php, программа, 564, 568
WEEK(), функция (MySQL), 82
WEEKDAY(), функция (MySQL), 82
while, циклы
построчная итерация по файлу, 521
Windows, системы
chr(), функция, 584
flock(), функция, 542
PEAR, инсталляция в, 609
SMTP-сервер, использование, 479
strftime(), функция
параметры, документация по, 77
символы форматирования, поддерживаемые в, 72, 76

ввод паролей в командной строке без визуализации, 584
загрузка и инсталляция PHP-GTK, 575
локали
изменение, 461
перечень доступных, 460
перенаправление вывода, 540
пути в, 509
разделители пути, 555
символы-ограничители строки, 509, 518, 519
сравнение с именами файлов, использование команды *dir* с `, 559
файлы, ограничители строк и пути, 509
функции, имеющие отношение к правам доступа и, 550
чтение двоичных файлов, 516
Window-target, заголовок, 203
wordwrap(), функция, 43, 457
wrap_html_tag(), функция, 163
_write(), метод (pc_DB_Session), 208
WSDL (Web Services Definition Language), язык определения веб-сервисов, 379
WWW-Authenticate, заголовок, 213

X

XHTML, преобразование символов возврата каретки в теги <br/>, 494
XML, язык разметки, 349, 359, 386
HTML и, 349
RSS-рассылки, чтение, 384, 386
SOAP-запросы, 376, 379, 382
XML-RPC, протокол, 351
вспомогательные функции PHP для, 369
клиенты, 375
посылка запросов, 369, 372
прием запросов, 372, 376
XSLT (eXtensible Stylesheet Language Transformation), язык преобразований XSL, 351, 366, 369
анализ с помощью DOM, 357, 360
на основе дерева, 358
анализ с помощью SAX, 360, 366
генерация вручную, 352
генерация с помощью DOM, 354, 357
создание нового документа, 355
узел верхнего уровня или корневой узел, 356
корневой элемент, 350
обмен данными с помощью WDDX, 382, 384
обмен сообщениями с сервером, 351
теги, 352
правила для, 349
XML_ATTRIBUTE_NODE, тип, 355
XML_ELEMENT_NODE, тип, 355
xml_parser_free(), функция, 363
XML_RSS::getChannelInfo(), функция, 385
XML_RSS::getItems(), функция, 385
XML_RSS::parse(), функция, 385
xml_set_character_data_handler(), функция, 362
xml_set_element_handler(), функция, 362
XML_TEXT_NODE, тип, 355
XML_Transform, пакет, 363
XML-RPC, расширение (см. XML), 369
xmlrpc_server_register_method(), функция, 373
xslt_close(), функция, 368
xslt_create(), функция, 367
xslt_errno(), функция, 368
xslt_error(), функция, 368
xslt_process(), функция, 367
XSLT-процессор Sablotron, 367
xu_rpc_http_concise(), функция, 371

Z

Zend Engine 2 (ZE2), интерпретатор, 182
именование конструкторов, 184
zip, расширение, 545
ZIP-архив, извлечение файлов из, 545
zlib, расширение, 543
zlib.output_compression, параметр конфигурации, 221
zlib.output_compression_level, параметр конфигурации, 221
zoneinfo, библиотека, 90

А

абзацы, подсчет в файле, 517
проблемы с, 518

адреса
IP, поиск с помощью DNS, 502
поиск с помощью LDAP, 497
адреса электронной почты
RFC 822-совместимый анализатор адресов в расширении IMAP, 400
комментарии, внедренные в, 399
поиск действительных с помощью регулярных выражений, 398
сокрытие от автоматов, отыскивающих адреса, 219
алгоритм Фишера-Йейтса для массивов, 524
алгоритм хеширования MD5, 411
анализ
DOM, 355
HTML, применение непоглощающего сравнения, 397
RSS-рассылки, 385
XML, 350
с помощью DOM, 357, 360
с помощью SAX, 360, 366
аргументов программы с помощью getopt(), 578, 581
аргументов, переданных программе в командной строке, 577
данные программы ping, 504
записи фиксированная ширина, в строках, 37
на основе дерева (DOM XML), 358
на основе событий, 361
файл протокола веб-сервера, 341
анонимное связывание (LDAP), 498
анонимный FTP, 495
арифметическое добавление или вычитание из даты, 87
ассоциативные массивы, 103
возвращаемые функцией getdate(), 67
возвращенные функцией pathinfo(), 555
итерация с циклами for по, повторное выполнение функции, 109
конфигурационные переменные в, 233
параметры функций в виде, 165
проверка на определенный элемент массива, 120
с целочисленными ключами, 113
удаление двойных элементов, 133
атрибуты
XML, 355
добавление к узлам DOM XML, 356
аутентификация
HTTP, базовая, 213, 215
PHP как CGI, 214
LDAP, применение для, 499
SMTP, 480
внедрение имени пользователя и пароля в URL, 514
личные вопросы пользователю, на которые атакующие не смогут ответить, 417
основанная на cookies, 216, 218

Б

базовые объекты, добавление, 194
базы данных, 273, 319
DBM, 280, 284
база данных имен пользователей и паролей, 281
доступ с помощью уровня абстракции DBA, 280
обработчики
различное поведение, 283
скомпилированные в вашей инсталляции PHP, 282
MySQL (см. MySQL), 276
Oracle (интерфейс OCI8), 275
PostgreSQL
DSN для, 285
применение обработчика, 209
SQL
выполнение запросов, 286
модификация данных в, 291
переносимость, 277
соединение с, 284
данные в последовательной форме, передаваемые в, 155
запросы, программное создание, 301, 305
извлечение строк без цикла, 288
кэширование запросов и результатов, 310, 312
нереляционные базы, поддерживаемые PHP, 273
определение количества строк, возвращенных запросом, 294
постоянные соединения, 277

постранично выведенные ссылки на множество записей, 309
постраничное отображение ссылок на множество записей, 305
преобразование кавычек в запрашиваемых данных в escape-последовательности, 295
простые текстовые файлы в качестве, 273
регистрация отладочной информации и ошибок, 297, 300
реляционные базы данных, поддерживаемые PHP, 273
сохранение сеансов в
pc_DB_Session, класс, 209
текстовые файлы в качестве, 279
уникальные идентификаторы, автоматическое присваивание, 300
уровень абстракции базы данных DB, 277
машины баз данных, поддерживаемые PEAR DB, 285
учетные записи пользователей, 240
хранение и извлечение сообщений, относящихся к различным темам, 312
хранение и извлечение сообщений, разделенных на потоки, 319
хранение сеансов в
pc_DB_Session, класс, 206
php_session, таблица, 206
эффективное повторение запросов, 292
базы данных DBM, 273, 280, 284
база данных имен пользователей и паролей, 281
обработчики, различное поведение, 283
обработчики, скомпилированные в вашей инсталляции PHP, 282
базы данных Oracle
DSN для, 285
машина базы данных OCI8
функции для подготовки и выполнения запросов, 294
базы данных, состоящие из плоских файлов (см. базы данных DBM), 273
безопасность
обработка форм, 265
работа с изображениями, 452
сокрытие сообщений PHP об ошибках от, 221
хранение cookie jar и, 328
шифрование и, 407, 431
SSL, 428
алгоритмы шифрования/дешифрования, 417, 423
ключи защищающего шифрования, 408
потерянные пароли, 416
проверка данных с помощью хеширования, 410
проверка надежности пароля, 414, 416
совместное использование зашифрованных данных с другим веб-сайтом, 426, 428
уровни шифрования, 408
хранение зашифрованных данных в файле или базе данных, 423
хранение паролей, 412
вне файлов сайта, 409
шифрование сообщений электронной почты с помощью GPG, 429
безымянные массивы, 106
бесконечно большие числа, 47
библиотека BCMath, 47
библиотека expat, 361
WDDX, использование с, 383
библиотека GD, 432, 457
(см. также графика), 435
версия 2.x, функции для дуг и эллипсов, 439
версия и конфигурация, проверка с помощью функции phpinfo(), 433
возможности различных версий, 432
координаты изображений, 435
форматы файлов, поддерживаемые, 432
шрифты, 441
библиотека GMP, 47
библиотека zoneinfo
часовые пояса, перечисленные, 91
бит смены идентификатора группы (setgid), 549, 552
бит смены идентификатора пользователя (setuid), 549, 552
блокировка файлов, 540, 543

база данных доски сообщений, разделенных на потоки, 318
базы данных DBM, 284
консультативная, 541
монопольная, 541
отмена, 542
текстовые файлы и базы данных, 280
файл как индикатор блокировки, 542
броузеры
PHP, действующий как, 320, 330
обнаружение различных, 209
конфигурируемые пользователем характеристики, например javascript или cookies, 210
свойства объекта, показывающие характеристики броузера, 210
передача выходной информации в, 218
передача изображений, 435, 452
прием сжатых ответов (Accept-Encoding, заголовок), 221
буферизованные данные, 159
вывод HTTP, смешивание заголовков и текста тела, 228

В

ввод/вывод (I/O), 507
двоичный код CLI, стандартные потоки I/O, 574
запись в стандартный поток вывода, 533
каналы, 511
передача входной информации в программу, 537
передача выходной информации в броузер, 218
сброс вывода в файл, 532
стандартный поток вывода программы, чтение, 537
чтение в/запись из сжатых файлов, 543
чтение данных из файлов, 510
чтение из стандартного потока ввода, 515, 582
веб-программирование, 198, 248
cookies, 198
перенаправление по другому адресу, 203
удаление, 202
установка, 200
чтение, 201
cookie-аутентификация, 216
автоматизация веб-процессов, 320, 348
анализ файла протокола веб-сервера, 341
выделение информации на веб-странице, 333
извлечение содержимого удаленных URL, 330
извлечение ссылок из HTML-файла, 334
обнаружение свежих, 345, 348
обнаружение устаревших ссылок, 343, 345
отладка цикла HTTP-запрос/ответ, 330, 333
преобразование ASCII в HTML, 336
удаление тегов HTML и PHP, 338
шаблоны Smarty, 339
аутентификация, основанная на cookies, 218
базовая аутентификация HTTP, 213, 215
взаимодействие внутри Apache, 235
идентификация различных броузеров, 209
обработка ошибок
пользовательский обработчик, использование, 225
устранение ошибок «headers already sent» (заголовки уже посланы), 227
отслеживание сеанса, 204
хранение сеансов в базе данных, 205
передача выходной информации в броузер, 218
переменные окружения
установка, 232
чтение, 231
проверка злоумышленных пользователей, 242, 248
программа для активации/деактивации учетных записей веб-сайта, 240, 242

профилирование программы, 237, 240
регистрация отладочной информации, 229
регистрация ошибок, 226
сжатие веб-вывода с помощью gzip, 220
сокрытие сообщений об ошибках от пользователей, 221
строка запроса GET, построение, 211
установка конфигурационных значений PHP, 235
чтение переменных конфигурации PHP, 233
веб-сервер Apache
browscap, файл, 210
httpd.conf, файл, установка переменных окружения, 232
пароли, хранящиеся в переменных окружения, защита, 409
веб-сервисы (см. службы Интернет), 379
вектор инициализации (IV), 425
в режимах шифрования/дешифрования, 418
передача вместе с зашифрованным текстом, 421
функции для создания, 422
вертикальный текст, рисование, 442
високосные года, 83
включаемые файлы
пробельный символ в, 228
включение переменных
HTML с включенными переменными, печать с помощью heredocs, 25
функции и выражения внутри строк, 34
владелец файла
время последнего изменения, 551
изменение, 553
суперпользователь, 554
внешние программы, запуск, 535
передача входной информации в, 537
внутренние ошибки
E_USER_NOTICE, 299
военно-морская обсерватория США, информация о часовых поясах, 89
возврат каретки
удаление из строк, 35
возвращение значений функциями
возвращение по ссылке, 169
несколько, из одной функции, 170
волшебные кавычки, 287
заключение в кавычки значений символов-заместителей и, 296
восьмеричные значения
\ddd, обозначающий, 394
права доступа к файлу, записанные в, преобразование в более легкие для чтения строки, 564
права доступа, переданные функции chmod(), 554
преобразование элемента mode массива информации о файле в, 552
строки в двойных кавычках, escape-последовательность, 24
вращение текста, 446
временные переменные
обмен значениями переменных без, 148
присваивание объектов, 186
временные файлы
непосредственная модификация файла без использования, 531, 532
создание, 513
время (см. даты и время), 65
время выполнения запросов в базу данных, 230
время доступа к файлам, 550
время истечения срока действия cookies, 200
в прошлом, для удаления cookies, 202
время модификации файлов, 516, 550
обновление с помощью функции touch(), 551
время последнего доступа к файлам, 550
время экономии дневного света, 65
добавление или вычитание из дат, 88
флаг DST, 70, 71
вспомогательные функции (PHP) для XML-RPC, 369
встроенная константа M_E, 56
вывод, 332
буферизация, 159

в броузер, 219
смешивание http-заголовков и текста тела, 228
стандартная ошибка и, 332
веб, сжатие с помощью gzip, 220
форматированной информации броузеру на основе вывода канала, 538
вывод на печать
UTC, время, 89
массив в виде HTML-таблицы с горизонтально расположенными столбцами, 141
массив с запятыми, 118
ошибки базы данных, 297
теги XML, расстановка вручную, 352
выделенный текст, использование тегов HTML <b> или <i>, 531
выпадающие меню на основе текущей даты, 271
выполнение запросов к базам данных
SQL, 286
без цикла, 288
кэширование запросов и результатов, 310, 312
определение количества строк, возвращенных, 294
программное создание запросов, 301, 305
символы-заместители в запросах, 289
эффективное повторение запросов, 292
выражения
в фигурных скобках { и }, оценка, 149
включение в строки, 34
высокоточное время, выработка, 94
выход
базовая аутентификация HTTP и, 214
принудительный выход пользователя по истечении фиксированного интервала времени, 214
вычитание из даты, 87

Г

гиперболические функции, 60
гистограммы, созданные по результатам голосования, 454, 457
глобальные переменные
COMMAND_LINE, 573
доступ внутри функций, 175
несколько значений, возвращаемых функцией, использование для, 171
горизонтальные столбцы в HTML-таблице, 142–144
градусы, 59
грады, 60
границы слова (\b оператор регулярного выражения), 334
графика, 432, 457
безопасная работа с изображениями, 452
динамические изображения, 449, 451
закрашенный прямоугольник, создание, 435
координаты изображений, 435
прозрачный цвет, 451
процесс генерации изображения, 434
рисование дуг, эллипсов и окружностей, 438
рисование линий, прямоугольников и многоугольников, 436, 438
рисование текста, 441
рисование узорными линиями, 440
рисование центрированного текста, 444, 449
PostScript Type 1, шрифты, 446
TrueType, шрифты, 448
шрифты, встроенные в GD, 447
создание гистограмм результатов голосования, 454, 457
удаление изображений, 436
формат файлов, поддерживаемый GD, 432
графический интерфейс пользователя (Graphical User Interface) (см. GUI), 575
группировка символов для сравнения, 389
групповые символы
SQL, заключение в кавычки в запросах, 296
замена имен в оболочке, 296
оболочка, сравнение с именами файлов, 558

Д

данные буфера
сброс в файл, 532
блокированные файлы, 541
данные, закодированные алгоритмом Base64, 428
даты и время, 65–100
microtime(), функция, 53, 94
время истечения срока действия cookies, 200
время модификации файлов, 516
время последнего изменения URL, 345
всеобщее скоординированное время (см. UTC), 65
вставка текущих даты и времени с помощью функции NOW(), 293
вывод на печать в определенном формате, 72
вывод на печать календаря на месяц с помощью функции pc_calendar(), 98, 100
выделение из строк, 84
выпадающие меню на основе текущей даты, 271
высокоточное время, выработка, 94
добавление или вычитание из даты, 87
кнопка, показывающая текущее время, 591, 595
локализация, 466
метки даты/времени файлов, 550
начальная дата для интервалов времени, 95
негригорианские календари, работа с, 96
определение времени выполнения программы, 237
определение дня недели, месяца, года или номера недели в году, 81
переход на летнее время (см. летнее время, переход на), 93
получение интервалов времени, 95
преобразование в метку времени UNIX и обратно, 70
принудительный выход пользователя по истечении фиксированного интервала времени, 214
проверка корректности дат, 82
программы определения времени для настройки производительности, 231
продолжительность FTP-соединения, 496
разность между двумя датами, 77
юлианское представление дат, 79
сортировка дат, 127
строка ISO 8601, возвращаемая приложением XML-RPC, 372
текущие, определение, 67
вывод на печать текущей даты в формате месяц/число/год, 69
форматирование для локалей, 459
часовые пояса, определение времени в различных, 88, 93
смещения между часовыми поясами, 89, 93
двоичные данные
преобразование в обычный текст, 410
сохранение в строках, 44, 46
числа с плавающей точкой, представленные как, 49
двоичные файлы
FTP_BINARY, параметр, 496
открытие в не-POSIX системах, 512
чтение в Windows, 516
двоичный код CGI, 574
двойные элементы, удаление из массивов, 133
декодирование данных
base64_decode(), 410
pc_decode(), функция, 257
utf8_decode(), функция, 475
делитель, наибольший общий, 62
дельта, 49
день недели, месяца или года, 81
дескрипторы файлов, 275, 507
для стандартного потока ошибок и стандартного потока вывода, 539
запись в несколько одновременно, 534
запись веб-страницы в файл, 323
десятичные числа, 47
форматирование суммы в долларах, 57
дешифрование данных, 427
программа get-crypt.php, 425

действия пользователя, реакция на, 590
диапазоны, целые, инициализация массива, 106
динамическая реализация объекта, 196
динамические изображения, 449, 451
  страница с кнопкой (в HTML), 450
динамические имена переменных, 149
динамические классы, 195
динамические функции, 177
дни, определение переменного количества в месяцах и годах для интервалов времени, 96
добавление к дате, 87
добавление одного массива к другому, 114
документация
  MySQL UNIX_TIMESTAMP(), функция, 79
  PEAR-классы, 620
  параметры функции strftime(), 77
  расширение PHP-GTK, 576
доменные имена, получение информации о, 505, 506
домены
  (см. также DNS), 474
  cookies, определение для, 200
  верхнего уровня в адресах электронной почты, 399
  сообщения gettext, 474
Доминус, Марк-Джейсон (Dominus, Mark-Jason), 139
дополнение строки, 39, 46
доска сообщений
  относящихся к различным темам, хранение, 312
  разделенных на потоки, хранение, 319
дочерние классы, 179
дуги, рисование, 438
Дейкстра, Эдсгер (Dijkstra, Edsger), 140

Е

европейские языки, символы для, 476

Ж

журналирование
  анализ файла протокола веб-сервера, 341
  изменение уровня регистрации ошибок в целях управления отображением ошибок, 222
  отладочная информация, 229
    и ошибки базы данных, 297, 300
  программные ошибки, 226
  просмотры страниц каждым пользователем, 236
  сброс вывода и, 533
  сообщения об ошибках (PHP), сокрытие от пользователей, 221

З

зависимости пакетов PEAR, проверка, 616
заголовки
  HTTP
    Accept-Language, 473
    Content-Length, 325
    Content-type, 353, 435, 453
      изображение PNG, передача броузеру, 435
      установка для XML, 353
    Cookie, 327
    Host, 331
    Location, 203, 323
      посылка других заголовков вместе с, 203
    Window-target, 203
    WWW-Authenticate, 213
    запрос, 330
    использование вместе с расширением cURL, 328
    получение содержимого URL с помощью, 328
    сообщение об ошибке /headersalreadysent/ (заголовки уже посланы), 227
  mail, 480
  сообщения новостных групп, 487
    References, заголовок, 489
    метаинформация о почтовых сообщениях, 491
загруженные файлы, обработка, 263
заключительные теги, PHP, 527
закрашенные изображения
  линии, прямоугольники и многоугольники, 435, 436, 437
  прямоугольники для полос на гистограмме, 457

эллипсы и окружности, 439
закрытие
XSLT-процессора вручную, 368
анализатора XML вручную, 363
закрытые ключи, 430
записи, подсчет в файле, 517
записи фиксированной ширины, анализ в строках, 37
запись
в несколько файловых дескрипторов одновременно, 534
в стандартный поток вывода, 533
в стандартный поток ошибок, 534
входная информация для внешней программы, 537
заполнение массива, 112
запросы
HTTP
записанные в протоколе доступа к веб-серверу, 342
метод GET, построение строки запроса, 211
отладка, 330
отсутствие памяти, 198
перенаправление, 323
SOAP
посылка, 376, 379
прием, 379, 382
XML-RPC
посылка, 369, 372
прием, 372, 376
о доменах, Whois, 505
злоумышленные пользователи, программа для проверки, 242, 248
значения по умолчанию
для переменных, 147
присваивание параметрам функции, 166
установка для параметров функции, 162
значения, возвращаемые функциями, 146
несколько, из одной функции, 171
ошибка, 173
пропуск определенных, 171
значения, разделенные запятой (CSV), 36
explode(), функция и, 37

И

идентификаторы сеанса, 204
связь с именами пользователей, 217
иерархия, класс, 178
имена
cookie, хранящий идентификатор сеанса, 205
DSN (имя источника данных), 206
LDAP, отмеченные и стандартные, 498
конструкторы, 184
переменные класса, 181
серверов доменных имен, 506
имена пользователей
анонимный, 495
база данных DBM, 281
исключение из паролей, 414
машины баз данных PEAR DB, 286
пути, разделение на составляющие, 554
пути, в UNIX и Windows, 509
имена файлов, сравнение с шаблоном, 558
имена хостов
для серверов новостей, 488
поиск в DNS, 502
именованные параметры функций, 165
имя источника данных (DSN), 206, 285
имя пользователя
OCILogin(), функция, 275
индексированные ссылки, 306
отображение, 308
индексные узлы, 547, 548
индексы
массив
не начинающийся с 0 (нуля), 104
повторное выполнение с помощью функции array_splice(), 111
файлы DBM, 283
интернационализация, 458, 476
каталоги сообщений, управление с помощью gettext, 474
локализация дат и времени, 466
локализация денежных значений, 467
локализация изображений, 470
локализация текстовых сообщений, 462

локаль по умолчанию, установка, 461
локаль, использование, 460
перечень допустимых локалей, 460
ресурсы локализации, управление, 472
чтение или запись символов Unicode, 475
интернет-службы, 369, 372, 376, 477, 478, 506
LDAP, аутентификация пользователей с помощью, 501
SOAP, посылка запросов, 376, 379
доменные имена, получение информации о, 505, 506
отправка сообщений в новостные группы Usenet, 486, 489
перемещение файлов с помощью FTP, 494, 497
поиск в DNS, 502
поиск адресов с помощью, 497
почта
MIME, 481
отправка, 478
чтение с помощью IMAP или POP3, 483
чтение сообщений новостных групп Usenet, 489, 494
интерполяция переменных
в строки в двойных кавычках, 484
интерпретатор, автоматический запуск, 572
интерфейс OCI8 (базы данных Oracle), 275
истинные значения переменных, 173
история команд, 583
источник данных (LDAP), 498
итерации
по массивам, 107, 110
начиная с конца, 522
по файлам в каталоге, 557
по файлу, строка за строкой, 521
функция, значение времени выполнения каждой, 238
Иудейский календарь, 96
преобразование в Юлианское представление дат, 98

К

кавычки
волшебные в PHP, 287
преобразование в escape-последовательности данных в запросах, 295
календари
gregoriantojd(), функция, 80
pc_calendar(), функция, вывод на печать месяца с помощью, 98, 100
unixtojd() jdtounix(), функции, 88
негригорианский, работа с, 96
функции преобразования для различных негригорианских календарей, 97
каналы, 511
вывод программы, 538
открытие в режиме r для чтения стандартного потока вывода, 539
открытие ко внешней программе и запись в, 537
каталог /tmp, хранение cookie в, 205
каталог для сообщений в различных локалях, 462
каталоги, 547, 571
site-search.php, программа, 568, 571
в качестве индикаторов монопольной блокировки, 542
имена файлов, разделение на составляющие, 554
имена файлов, соответствующих шаблону, 558
информация о файле, получение, 551
копирование или перемещение файла, 556
метки даты/времени файла, получение и установка, 550
обработка всех файлов в, 557, 559, 561
права доступа к файлам или владельца, изменение, 553
программа, предоставляющая список файлов веб-сервера, 564, 568
создание нового, 561
удаление со всем содержимым, 563
удаление файлов, 556
каталоги сообщений, 462
catalog-compare.php, программа, 472
strftime(), функция строки формата как сообщения, 466
создание, управление и работа с ними посредством gettext, 474

классы, 178, 182
Cache_DB, 310
DB_Sql в PHPLib, 286
pc_RSS_item, 363
pc_RSS_parser, 363
pc_ в именах, 20
SOAP_Client, 377
SOAP_Server, 380
SOAP_Value, 378
SOAP_WSDL, 379
XML_RSS, 384
динамическое создание, 195
иерархия, 178
каталога, 547
методы, 178
определение и создание, 179
ошибки базы данных, 297
расширение, 181
свойства, 178
символьный, 389
создание подкласса или расширение, 179
клиенты
FTP, 494
LDAP, 497
SOAP, 376
XML-RPC, 375
основанные на веб, для электронной почты, 483
клонирование объектов, 185
ключевые слова
class, 179
extends, 181
function, 179
new, 179
реализация объектов, 182
parent::, 188
var, 179
функция, 160
ключи, массив
ассоциативные и с числовой индексацией, 101
проверка массивов на определенные, 119
соединенные массивы и, 115
строки как, 172
уникальность, 106
хранение нескольких элементов с одним ключом, 105
числовые, отрицательные числа в, 105
кнопки, 449
GtkButton, класс, 586
кодирование
ASCII, 475
base64, 410
HTML, элементы, 336
pc_encode(), функция, 257
Unicode, 475
URL, 211
UTF-8 (Unicode), 476
строки формата для, 44
элементы HTML
экземпляр RSS в виде HTML, 364
коды языков, 463
командная строка, включение внешних данных в, 535
команды
dir (Windows), 558
ls (UNIX)
оператор ‘ (обратный апостроф), использование с, 558
PEAR, перечень всех допустимых, 610
PHPDoc, 620
REPLACE INTO (MySQL), 208
история, 583
комбинации элементов массива, определение всех, 136, 138
комментарии
внедренные в адреса электронной почты, 399
отладка, уровни приоритета для, 230
конвертация массивов, 120
константы, 146
M_E, 56
M_PI, 60
присваивание постоянных значений свойствам класса, 179
размещение в левой части сравнения, 147
режимы шифрования, 422
конструкторы, 180
класса, 180
объект, вызов родительского, 188
объект, определение, 183
консультативная блокировка файлов, 541
контейнеры для GUI-элементов управления окном, 587, 590

GtkHBox, 588
GtkTable, 589
GtkVBox, 588
панель меню, 593
конфигурационные переменные, PHP, 233
конфигурационные файлы
пространства имен и, 527
чтение, 527
координаты, графические изображения, 435
дуги, 438
линии, прямоугольники, и многоугольники, 437
центрированный текст в изображении, 446
центрированный текст в шрифтах PostScript Type 1, 446
центрированный текст в шрифтах TrueType, 448
центрированный текст в шрифтах, встроенных в GD, 448
копирование файлов, 556
корневой узел (DOM XML), 356
корневой элемент (XML), 350
косеканс, 59
котангенс, 59
криптография (см. шифрование), 422
кэширование
запросов к базам данных и результатов, 310, 312
результаты системного вызова stat() и функции, предоставляющей информацию о файле, 553
кэшированные данные, контейнер для, 312

Л

летнее время (см. время экономии дневного света), 65
летнее время, переход на, 93
лимиты времени
веб-серверы, ожидание запросов, 330
линии, рисование
пунктирная, 452
узорные, 440
липкий бит, 549
логарифмы, 55
по основанию 10, 55
по основанию e, 55
логические значения переменных false и true, 145
логические константы и значения, возвращаемые функциями, 146
ложные значения, 173
локали, 458
использование определенной, 460
каталоги для изображений, 470
перечень допустимых, 460
по умолчанию, установка, 461
языки, коды стран, и спецификаторы наборов символов, 459
локализация, 458
включаемых файлов, 471
даты и время, 466
каталоги сообщений, управление с помощью gettext, 474
текст с изображениями, 470
текстовые сообщения, 462
управление ресурсами, 472
форматы денежных единиц, 467
локальная область видимости, 176

М

максимальное сравнение (см. поглощающее сравнение), 396
мантисса, 49
маска (umask), установка прав доступа, 549
массивы, 101, 144
$_COOKIE, 201
$_ENV, 231
$_FILES, 263
$_SESSION, 204
{}, разрешение неопределенности с помощью, 150
включение элементов в строки, 34
возвращаемые функцией localtime(), 68
вывод на печать HTML-таблицы с горизонтально расположенными столбцами, 141
выполнение цикла foreach по, 103
добавление одного к другому, 114
замена ключей и значений, 120
изменение длины, 112
заполнение, 112
индекс, не начинающийся с нуля, 104

инициализация массива диапазоном целых чисел, 106
информация о файле, возвращенная функцией stat(), 551
выполнение итерации по, 553
итерация по, 110
с помощью функции each(), 172
массивов, 103, 137
нахождение элемента с наибольшим или наименьшим значением в, 123
нахождение элементов, удовлетворяющих определенным требованиям, 122
несколько значений, возвращаемых функцией, 170
объектов, 103
определение всех комбинаций элементов, 136, 138
определение всех перестановок, 139, 141
определение объединения, пересечения, или разности между, 134
определение позиции элемента, 121
определение со строковыми ключами, 103
ошибки формы, хранение в, 259
перебор, 107
передача в функции с переменным количеством аргументов, 167
передача в функции с переменными аргументами, 169
переменные класса, сохраненные в, 190
переменные, принудительное преобразование в форму массива, 109
преобразование в строки, 116
проверка наличия ключа, 119
проверка наличия элемента, 119
проверка с помощью функции is_array(), 109
рандомизация порядка элементов, 131
реверсирование порядка элементов, 124
режимы выборки запросов к базам данных, 288–290
с числовой индексацией, 101
сортировка, 125
метод, использование вместо функции, 130
множественная, 129
тасование колоды карт, 132
строковое представление, 154
типы данных элементов, 102
удаление двойных элементов из, 133
удаление элементов из, 110, 112
файлов, возвращенных функцией pc_process_dir(), 560
хранение нескольких элементов с одним ключом, 105
числовые, 135
удаление двойных элементов, 133
чтение конфигурационного файла в, 526
чтение файла в, 521
чтение файла в и удаление последнего элемента, 529
чтение файла в с последующей перетасовкой строк, 524
чтение файла с последующим обращением, 522
элементы, разделенные запятыми, вывод на печать, 118
машина базы данных Interbase, 294
менеджер пакетов, PEAR (см. pear, программа), 607
меню
показ в окне GTK, 592, 595
создание выпадающих на основе текущей даты, 271
метасимволы, 388, 390
\d (цифра) в регулярных выражениях, 390
\s (пробельный символ), 390
URL, преобразование в escape-последовательности, 154
\w (слово), 390, 394
преобразование в escape-последовательности в Perl-совместимых регулярных выражениях, 404
среда, преобразование в escape-последовательности во внешние файлы, 535
метка конца строки, 25
метки времени UNIX, 66
localtime(), функция, 69
время истечения срока действия cookies и, 200
определение разности между, 77

преобразование в понятный человеку формат, 493
преобразование в юлианское представление дат и обратно, 88
преобразование понятных человеку строк даты и времени в, 85
метки даты/времени, получение и установка, 550
метки для кнопок, установка текста для, 592
методы, 178
(см. также функции), 378
GET, 330
перенаправление пользователя по другому URL, 203
получение содержимого URL с помощью, 322
построение строки запроса, 211
_ _get() и _ _set(), 189
POST, 331
SOAP-запросы, 381
URL перенаправление и, 203
XML-RPC запросы, 373
получение содержимого URL с помощью, 324, 326
регистрационная форма для LDAP, 500
XML-RPC
отличия от PHP в именовании, 370
связь с сервером и функциями PHP, 374
вызов для объекта, возвращенного другим методом, 186
класс
parent:: перед именем метода, 182
наследование дочерними классами, 181
конструкторы
класс, 180
определение объектов, 183
объект
динамическая реализация объекта и, 195
обнаружение, 192
профилирование выполнения, 240
переопределение, доступ, 187
полиморфизм, 190, 192
пространства имен и, 378
сортировка массивов, 130
минимальное сравнение (см. непоглощающее сравнение), 396
многопотоковые системы, высокоточное время и, 95
многоугольники, рисование
закрашенных, 437
незакрашенных , 436
модификаторы шаблонов, 390, 392
модификация файлов, время, 550
модификация метаданных файлов, 550
модуль сеанса, 204
наблюдение за пользователями с помощью, 236
монопольные блокировки, 541

Н

наборы символов
в синонимах локалей, 461
определение для локалей, 459
наибольший общий делитель (НОД), 62
наименьший значащий бит, 476
наследование, класс, 179
реализация путем расширения классов, 181
настройка производительности, 230
национальные настройки
определения слова и, 394
начальное значение для генерации случайного числа, 53
начальные теги, PHP, 527
начальный и конечный углы (в градусах) дуг, 438
недели
стандарт ISO для, 82
стандарт ODBC, 82
немедленное удаление объекта, 184
непоглощающее сравнение, 395
нереляционные базы, поддерживаемые PHP, 273
несколько измерений внутри одного массива, сортировка, 129
несколько массивов, одновременная сортировка, 129
новостные группы Usenet, 486
отправка сообщений в, 486, 489
чтение сообщений, 489, 494
новые строки
строки в двойных кавычках, 24
номер недели в году, 81

нулевые символы
удаление из строк, 35

О

область видимости, переменные, 176
обновление пакетов PEAR, 618
оболочки
универсализация имен файлов, 558
обработка ошибок
настройка путем изменения уровня регистрации ошибок, 222
ошибки HTML, запрет, 235
пользовательский обработчик ошибок, использование, 225
регистрация программных ошибок, 226
константы _ _FILE_ _ и _ _LINE_ _, 227
типы ошибок, перечень, 224
уровни отчета, 223
устранение ошибок «headers already sent» (заголовки уже посланы), 227
объединение массивов, 134, 135
объект родительского узла, 359
объектно-ориентированное программирование, объекты PHP, 179
объектно-ориентированный интерфейс к информации каталога, 557
объекты, 182, 195
GTK, 586
PHP, 179
SOAP, 377
базовый, добавление свойств к, 194
возвращенный другим методом, вызов метода для, 186
для каталогов сообщений, 464
клонирование, 185
массивы, 102
метод-конструктор, определение, 183
методы и свойства, обнаружение, 192
модель объекта Zend Engine 2 (ZE2), 182
реализация, 179
динамическая, 196
свойства
включение в строки, 34
характеристики броузера, 210
создание, 178
ссылки на, 185
строковое представление, 154
удаление, 184
узлы DOM, 359
языки, хранящиеся в, 472
объявление/определение
классы, 179
переменные класса, 179
свойства класса, 179
статические переменные, 150
функции, 160
одностороннее шифрование, 407
crypt(), функция, использование в, 413
окна
показ нескольких GUI-элементов управления окном, 587, 590
показ панели меню, 592, 595
создание в PHP на стороне клиента, 586
округление чисел с плавающей точкой, 50
окружности, рисование
закрашенных, 439
операторные дескрипторы, 275
операторы
конкатенации строк, 34
соединение массивов, 115
логического НЕТ (~), 223
обратного апострофа (‘), 538, 558
перенаправления, 539
присваивания, 169, 186
равенства (==), 120, 146
тождества (===), 146, 147
тождественного неравенства (!==), 122, 557
трехчленный (?:), 147
определение альтернатив при сравнении с шаблоном-регулярным выражением, 390
организация (LDAP), 498
ответы, HTTP
отладка, 330
отладка
проблемы базы данных, 297, 300
регистрация информации, 229
регистрация ошибок, использование в, 226
цикл HTTP-запрос/ответ, 330, 333
отмена блокировок файлов, 542

отрицательные числа в ключах массивов, 105
отслеживание сеанса, 204
хранение сеансов в базе данных, 205
pc_DB_Session, класс, 206, 209
ошибка, возвращенная функциями, 173
ошибки
Getopt_Error, класс, 581
база данных, журналирование, 297, 300
запись в стандартный поток ошибок, 534
ошибочное написание имен параметров, 165
регистрация, 175
стандартный, 539
уловимые, 224
функции с одинаковыми именами, 160

П

пакеты, PEAR
PHP Extension Code Library, библиотека дополнительных программ PHP (PECL), 607
PHP Foundation Classes, основные классы PHP, 607
нахождение, 612
обновление, 618
удаление, 619
память, совместно используемые сегменты, 152
параметры функции, 161
доступ, 161
именованные, 165
передача по ссылке , 164
переменное количество, 167, 169
установка значений по умолчанию, 162
пароли
URL, защищенные с помощью, 427
анонимный FTP, 495
аутентификация HTTP, 213
база данных DBM, 281
машины баз данных PEAR DB, 286
потерянные, работа с, 416
проверка надежности, 414, 416
хранение вне файлов сайта, 409
чтение из командной строки без визуализации, 583, 586
шифрование и хранение, 412
пары ключ/значение для баз данных DBM, 282
пары ключ/значение, массив
замена ключей и значений с помощью функции array_flip(), 120
определение с помощью оператора =>, 103
связь между, сохранение при сортировках, 128
сохранение связей при сортировке массива между, 125
первая неделя в году, 82
перегрузка свойств, 189, 190
передача по значению, 162
передача по ссылке, 164
несколько значений, возвращенных функцией в массив, 171
параметры функции, 162
переменные, 145, 159
PHP_AUTH_USER и PHP_AUTH_PW, 213
включение в строки, 34
возвращение по ссылке, 169
временные, присваивание объектов, 186
значение по умолчанию, установка, 147
интерполяция в строки двойных кавычках, 484
истинные значения, 173
класс, 179
имена свойств, 181
присваивание не постоянного значения, 180
сохраненные в массиве, 190
не определенные, обработка ошибок и, 223
обмен значениями без временных переменных, 148
объект, обнаружение, 193
параметры функции, 161
передача по значению или по ссылке, 162
передача по ссылке, 164
переменные окружения, 231
получение дампа содержимого в виде, 156

получение дампа содержимого в виде строк, 159
принудительное преобразование в форму массива, 109
пустые, оценка как логического значения false, 145
сложные типы данных, инкапсуляция в строках, 154
совместное использование процессами, 152, 154
создание динамического имени, 149
сравнение, ошибки при использовании операторов = и ==, 146
статические, 150
установленные и сброшенные, 145
чтение XML с помощью DOM, 357
переменные и их значения, преобразование в текстовую форму, 154
переменные окружения
HOME и USER, установка для GPG, 430
HTTP_USER_AGENT, 210
LC_ALL, 462
ORACLE_SID, 275
установка, 232
хранение паролей в, 409
чтение, 231
переменные переменных, 35
вызов различных функций, в зависимости от значения переменной, 174
имена глобальных переменных, определение, 176
итерация по переменным с похожими именами, 150
создание динамических имен переменных, 149
установка локальных значений внутри функций, 148
перемещение файлов, 556
перемещение элементов массива, 112
перенаправление
cookies по другому адресу, 203
HTTP-запросы, 323
ввод/вывод
стандартный поток ошибок в стандартный поток вывода, 539
переопределенные методы, доступ, 187
пересечение массивов, 134
array_intersection(), функция, 135
перестановка массивов Фишера-Йетса, 132
перестановки массива, 139, 141
переход на летнее время
разности между двумя метками времени, влияние на, 78
разность между двумя датами с помощью юлианского представления дат, 80
часовые пояса zoneinfo, информация о, 91
печать текста на испанском языке, 465
плотность расположения букв, 446
текстовая графика шрифта PS, 443
побайтное представление символов в кодировке UTF-8, 476
поглощающее сравнение, 395
поддержка GIF, удалена из GD, 432
подкласс, создание, 179
подстроки (см. строки), 26
подсчет строк, абзацев или записей в, 517
позиция
получение текущей в файле, 528
элементы массива, определение, 121
поиск
site-search.php, программа, 568, 571
полиморфизм методов, 190, 192
получение дампа содержимого переменных в виде строк, 156, 159
поля
скрытые поля формы, хеши в, 411
текстовые поля переменной длины, обработка в файле, 524
пользователи
злоумышленные, отсечение, 242, 248
реакция на, 590
порядки, 49, 56
последняя строка в файле, удаление, 529, 530
постоянные соединения с базами данных, 277
закрытие для DBM, 282
постраничное отображение результатов запроса PEAR DB, 305, 309
почта
MIME, 481
отправка, 478
с помощью внешних программ, 479

чтение с помощью IMAP или POP3, 483
пояса часовые, 65
права доступа, 512, 549
mode, элемент массива информации о файле, преобразование в восьмеричное представление, 552
биты смены идентификатора пользователя (setuid), биты смены идентификатора группы (биты setgid и sticky), 549
время последнего изменения, 551
значения для, 549
изменение, 553
преобразование восьмеричных представлений в более легкие для чтения строки, 564
семафоры для совместно используемых сегментов памяти, 152
суперпользователь, 554
права доступа группы, 549, 552
изменение владельца файлов, 554
права доступа пользователя, 549, 552
право владения файлами, 549
право на выполнение, 549
право на запись, 549
право на чтение, 549
превращение в escape-последовательности
метасимволы в регулярных выражениях, 404
превращение всех квантификаторов из поглощающих в непоглощающие с помощью модификатора U, 396–397
предупреждения, подавление в GPG, 430
преобразование XML-документов с помощью XSLT, 366, 369
преобразование в escape-последовательности, 535
HTML, 336
SQL и групповые символы для замены имен в оболочке, 296
данных, введенных пользователем в HTML-странице, 267
кавычек в данных запроса, 295
ограничители шаблона-регулярного выражения, 392
элементов HTML, 212
привязки при сравнении с шаблонами-регулярными выражениями, 389
принуждение PEAR к инсталляции новой версии, 620
приращение при инициализации массива диапазоном целых чисел, 107
присваивание
=&, оператор, 169, 186
= оператор, 146
= и =&, операторы, 170
функция возвращает переменным, 160
пробельный символ, 30
\s, метасимвол в регулярных выражениях, 390, 520
в HTML и XML, 350
во включаемых файлах, 228
замена на табуляции и обратно в строках, 30
пробел в шрифте, 443, 446
удаление завершающего, 509
удаление из строк, 35
проверка корректности
входная информация формы, 253
даты, 82
программы
get-crypt.php, 425
gtk-weather.php, 598
unzip.php, 545
профилирование, 237, 240
чтение стандартного потока вывода из, 537
чтение стандартного потока ошибок из, 539
продолжительность, применительно к FTP-соединениям, 496
прозрачности, 433
прозрачный цвет, 451
простая разность массивов, 135
простой API для XML (Simple API for XML) (см. SAX), 361
пространства имен
SOAP, 378, 380
новые возможности PHP, 527
проталкивание нового значения на вершину стека, 102
протоколы, 477
FTP, 477
прототип функции
доступ к параметрам функции, 161
присваивание значений по умолчанию параметрам, 162
профилирование программы, 237, 240

процессы, совместное использование переменных, 152, 154
прямоугольники, рисование, 436
закрашенных, 436
полосы на гистограмме, 457
псевдоконструкторы, 183
пустое множество, 137
пустые переменные, не установленные переменные как, 145
пустые строки
присваивание параметру функции в качестве значения по умолчанию, 163
присваивание элементам массива, 110
сообщения об ошибках, 164
пути, 554
для cookies, 200

Р

радианы, 59
разделители шаблона, 388
разделитель аргументов, замена & (амперсанда) на ; (точку с запятой), 212
разделяемые блокировки, 541
разности массивов, 134
array_diff(), функция, получение простой разности, 135
симметрическая разность, 136
создание своего собственного алгоритма для, 135
рандомизация
всех строк в файле, 523
массивы, 131
расширения
cURL, 321
извлечение содержимого удаленных URL с помощью, 327
DOM XML, 354
анализ XML, 357
FTP, 494, 497
GD, 432, 457
(см. также GD), 432
gettext, 474
IMAP, 398, 483
RFC 822-совместимый анализатор адресов, 400
LDAP, 497
mcrypt, 417
PHP-GTK, 572, 576, 586
запуск программ с помощью интерпретатора CLI, 575
запуск программ в режиме CGI, 575
Readline, 582
WDDX, 382
XML, 349, 386
XML-RPC, 369
xmlrpc-epi, 369
XSLT, 366
zip, 545
zlib, 543
классов, 179
получение переменных конфигурации для, 234
функции шифрования, 407
расширенные регулярные выражения (см. функции ereg, регулярные выражения), 388
расшифровка данных
(*см. также* шифрование), 418
mcrypt_decrypt(), функция, 418
реализация объектов, 182
new, ключевое слово
использование, 179
динамическая, 196
реверсирование массивов, 124
регистр
изменение в строках, 32
установки в локалях, 458
модификатор шаблона i (не чувствительный к регистру), 391
сравнение, не чувствительное к eregi(), функция, 392
сравнение, чувствительное к, 40
чувствительность к
в heredocs, 25
в XML, 363
в функции array_diff(), функция, 135
регулярные выражения, 387, 406
\s (пробельный символ) метасимвол, 520
HTML-строка, поиск тега изображения, 388
pc_link_extractor(), функция
использование, 335
Perl-совместимые, функции preg для, 388

POSIX и Perl-совместимые, информация веб-сайта о, 391
preg_match(), функция и, 333
сравнение строк в комбинированном формате протокола (Combined Log Format), 341
в качестве разделителей при разбиении строк, 42
выделение дат и времени из строк, 86
выделение текста, заключенного в теги HTML, 402
для расширений файлов, 192
метасимволы в, 388, 390
экранирование, 404
модификаторы шаблона, 390
нахождение n-го совпадения, 394
нахождение всех строк в файле, соответствующих шаблону, 401, 404
переход от функций ereg к функциям preg, 391, 393
ограничители шаблона в preg, 392
поглощающее и непоглощающее сравнение, 395
поиск действительных адресов электронной почты, 398
поиск слов, 393
поиск шаблона-разделителя для записей, 405
преобразование текстовых адресов электронной почты в гиперссылки mailto:, 387
совпадение ограничителей строк в UNIX и Windows, 518, 519
сравнение имен файлов с шаблоном, 558
условия поиска для программы site-search.php, 568
эквивалент функции trim(), 390
режимы
права доступа к файлу, 512
преобразование восьмеричных представлений в более легкие для чтения строки, 564
элемент mode массива информации о файле, 552
шифрование, 425
режим CBC (Cipher Block Chaining) цепочки кодовых блоков, 421
режим CFB (Cipher Feedback) кодовой обратной связи, 421
режим ECB (Electronic Code Book) электронной кодовой книги, 421
режим OFB (Output Feedback) выходной обратной связи, 421
векторы инициализации при, 421
результаты голосования, гистограммы, созданные по, 454, 457
рекурсивные функции, 139
обработка узлов DOM XML-документа с помощью рекурсии типа сначала вглубь, 358
получение дампа содержимого в виде строк, 159
получение дампа содержимого переменных в виде строк, 156
реляционные базы данных, поддерживаемые PHP, 273
рисование
дуги, эллипсы и окружности, 438
линии, прямоугольники и многоугольники, 436, 438
текст в графическом виде, 441
узорные линии, использование, 440
центрированный текст, 444, 449
PostScript Type 1, шрифты, 446
TrueType, шрифты, 448
шрифты, встроенные в GD, 447
родительские классы, 179
идентификация, 188
родительский каталог, создание, 562
руководство по PEAR, веб-сайт, 609

С

сброс данных из буфера в файл, 532
свойства
класс, 178
именование, 181
объявление, 179
присваивание непостоянного значения, 180
присваивание постоянных значений, 179
объект, 192

добавление к базовому объекту, 194
перегрузка, 189, 190
связывание (с LDAP-сервером), 498
сглаживание, текстовая графика шрифта PS, 443
сдандартные имена (LDAP), 500
секанс, 59
семафоры
обеспечение монопольного доступа к разделяемой памяти, 152
серверы
DNS, выделение имен и IP-адресов, 506
IMAP, 484
LDAP
взаимодействие с, 498
загрузка, 498
POP3, 484
SMTP, 479
SOAP_Server, класс, 380
XML-RPC, 373
анализ файла протокола, 341
запись сообщений PHP об ошибках в журнал, 221
лимиты времени на ожидание запросов, 330
прокси-сервер HTTP, конфигурация PEAR для использования с, 611
расшифрованные данные, подглядывание, 426
список содержимого каталога веб-сервера (web-ls.php), 568
получение списка, 564
сериализация WDDX-переменных, 383
сеточная схема размещения элементов управления окном, 589
сжатие слов и букв, 443
сжатые файлы
gzip, для веб-вывода, 220
извлечение из ZIP-архива, 545
чтение и запись, 543
сигналы
от GUI-элементов управления окном, обработка, 586
от элементов меню в окнах GTK, 593
связь с функциями обратного вызова для обработки, 590
символические ссылки, 548
is_dir(), функция и, 561
вызов функции stat() для, 553
символы
NUL, открытие двоичных файлов и, 512
базовые (salt), добавленные в зашифрованные пароли, 413
возврата каретки, 509
преобразование в теги XHTML <br/>, 494
обработка в строке по отдельности, 28
новой строки
\n \n, задание двойного интервала, 43
в Windows, 509
в функциях-обработчиках файлов, 509
переход от функций ereg к функциям preg, 393
удаление из строк, 35
переворот в строках, 30
символы-заместители в запросах к базам данных, 289, 292
символьный класс
инвертирующее или совпадающее дополнение, 389
симметрическая разность массивов, 134, 136
синонимы
_() для функции gettext(), 475
для распространенных локалей, 461
система шаблонов (Smarty), 339
показ строк, извлеченных из базы данных, 339
системные вызовы
stat(), 553
pc_mktime(), функция, 90
показываемые функцией strace(), перехват, 539
системы Windows
dir, команда, 558
Event Log в Windows NT, 222
перенаправление стандартного вывода, 539
разделяемая память и, 154
скалярные переменные, обработка массива и, 109
скрытые поля формы, 251
слежение за сеансом с помощью, 256

уникальный идентификатор в, 261
хеши в, 411
слова
\w (слово), метасимвол, 390
обработка всего файла, 519
образование множественного числа, 57
переворот в строках, 30
преобразованиепервого символа слов в строках в верхний регистр, 33
сравнение с помощью регулярных выражений, 393
слова во множественном числе, 57
слова из словаря, исключение из паролей, 414
словарные атаки, 413
службы Интернета (см. интернет-службы)
случайные числа
вычисление с помощью функции pc_randomint(), 523
генерация в пределах диапазона, 52
источники для векторов инициализации, 422
со смещением, генерация, 54
смещения между часовыми поясами, 89, 93
изменение жестко запрограммированных для DST, 93
совместно используемые сегменты
хранение разделяемых переменных в, 152
соединение массивов, 114
чтобы найти объединение, 135
сокеты
SSL (Secure Sockets Layer), уровень защищенных сокетов, 201, 329, 484
открытие с помощью fsockopen(), 320
соединения с серверами новостей, 487
сообщения
локализация, 462
относящиеся к различным темам, хранение и извлечение, 312
отправка в новостные группы Usenet, 486, 489
разделенные на потоки, хранение и извлечение, 319
форматирование текстовых сообщений для локалей, 459
чтение новостных групп Usenet, 489, 494
сообщения об ошибках
по умолчанию или пустые, 164
сокрытие от пользователей, 221
сортировка
массивы, 125, 130
в порядке убывания, 124
изменение типа сортировки, 130
множество, 129
по вычисляемому полю, 126, 129
предварительное копирование для сохранения исходного порядка, 124
тасование колоды карт, 132
текст для различных локалей, 458
сохранение данных без их распечатки, 159
специальные символы
SQL и групповые символы для замены имен в оболочке, преобразование в escape-последовательности, 296
в SQL, 278
в кодировании URL, 212
в регулярных выражениях (см. метасимволы), 388
превращения в escape-последовательности в HTML, 268
преобразование в escape-последовательности в запрашиваемых данных, 295
спецификация W3C DOM, 359
список слов, используемых при словарной проверке, 414
сравнения, константы в левой части, 147
среда, метасимволы, преобразование в escape-последовательности, 535
ссылки
извлечение из HTML-файла, 334
на объекты
окна GTK, 586
присваивание, 185
на разделяемые сегменты памяти, 153
постранично выведенные, множество записей базы данных, 305, 309
свежие, обнаружение, 345, 348

устаревшие, обнаружение, 343, 345
стандарт ODBC
недели года, 82
поддержка в PHP, 273
стандартная ошибка
вывод отладочной информации модуля cURL в, 332
стандартные имена (LDAP), 498
стандартный поток ввода, 574
чтение из, 515
чтение с клавиатуры, 582
стандартный поток вывода, 574
запись в, 533
перенаправление стандартного потока ошибок в, 539
чтение из программы, 537
стандартный поток ошибок, 574
запись в, 534
чтение из программы, 539
статические переменные, 150
страна (LDAP), 498
строки, 23, 46
анализ данных, разделенных запятой, 36
анализ записей фиксированной ширины в, 37
в двойных кавычках, интерполяция переменных в, 484
включение функций и выражений, 34
во встроенных документах (heredocs), 24
возвращение всего вывода программы в одной строке, 538
выбор случайной строки из файла, 522
вывод DOM XML-документа в, 357
выделение дат и времени из, 84
выделение регулярного выражения, 405
двоичные данные, сохранение в, 44, 46
инициализация, 23
как ключи массива, 103
массивы для нескольких возвращаемых значений, 172
смешивание с числовыми, 105
комбинированный формат протокола (Combined Log Format) NCSA, анализ, 341
нахождение всех строк, соответствующих шаблону, 401
переворот пословно или посимвольно, 30
подстроки
доступ, 26
замещение, 27
получение дампа содержимого переменных в виде, 156, 159
посимвольная обработка, 28
преобразование в числа и обратно, 47
преобразование массивов в, 116
проверка правильности записи числа, 48
пустые, 145
присваивание параметру функции в качестве значения по умолчанию, 163
разбиение на части, 40, 42
рандомизация всех строк в файле, 523
расширение и сжатие табуляций, 30, 32
регистр, управление, 32
сжатие, 544
сложные типы данных, инкапсуляция в, 154
serializing strings, 154
удаление последней строки из файла, 529, 530
удаление пробелов из, 35
упаковка текста, 42
числовые, 48
чтение определенной строки в файле, 521
чтение файлов в, 515
строки, дополненные пробелами, 39, 46
структурированный доступ к базам данных, 280
суперглобальные массивы
$_COOKIE, 249
$_ENV, 249
$_FILES, 249
$_GET, 249
$_POST, 251, 266
$_REQUEST, 249
$_SERVER, 249
$_SESSION, 217, 256

суперпользователь, изменение владельца и прав доступа к файлу, 554
сценарии
CGI, PHP как, 214
функциональность GTK, загрузка с помощью d1(), функция, 576

Т

табуляция
ASCII-представление в шаблоне или в замещающих значениях ereg, 393
\t, в строках в двойных кавычках, 24, 393
расширение и сжатие в строках, 30, 32
удаление из строк, 35
тасование колоды карт, 132
теги

 (XHTML), преобразование символов возврата каретки, 494
HTML, 338
<b> или <i> для выделенного текста, 531
выделение текста, заключенного в, 402
PHP, начальные и заключительные, 527
PHPDoc, 620
XML
выполнение цикла по данным и вывод на печать, 352
текст
адреса электронной почты, преобразование в mailto: гиперссылки, 387
выделение заключенного в теги HTML, 402
выделенный, 531
каталоги сообщений для локалей, применение gettext, 474
локализация сообщений, 462
локализованный, вывод вместе с изображениями, 470
преобразование двоичных данных в, 410
рисование в графическом виде, 441
вертикально, 442
программа вывода гистограммы, 456
центрированный, 444, 449
шрифты, 441
рисование с помощью библиотеки GD, 433
сортировка для различных локалей, 458
узлы XML, 355
упаковка в строку определенной длины, 42
текстовые поля (переменной длины), обработка в файле, 524
текстовые файлы как базы данных, 273, 279
структурированный доступ, блокировка в, 280
текущая позиция в файле, 528
тело сообщений
новостных групп, 488
электронной почты, 485
терминалы, управление характеристиками в UNIX, 584
типы MIME, 481
значения IMAP, 486
типы данных
комплексный, инкапсуляция в строках, 154
преобразование с помощью функции unpack(), 46
проверка чисел на принадлежность к определенному типу, 49
элементы массива, 102
точность
числа с плавающей точкой, округление до ближайшего целого, 50
трехчленный (?:) оператор, 147
тригонометрические функции, 59
в градусах, 60

У

уведомления, сообщения об ошибках, отмеченные как, 223
угол, вращение текста, 443, 446
удаление, 563
(см. также перемещение), 202
cookies, 202
двойных элементов из массива, 133
изображений, 457
каталога и его содержимого, 563
пакетов PEAR, 619
первого и последнего элементов массива, 112

последнего элемента из массива и возвращение его, 102
последней строки из файла, 529, 530
символов-ограничителей строк, 509
файлов, 556
элементов массива, 110, 112
удаленные файлы, открытие, 514
узел верхнего уровня, 356
узлы
DOM XML
верхнего уровня или корневой, 356
добавление атрибутов в, 356
создание и добавление, 356
XML, 355
узорные линии, рисование, 440
уникальные идентификаторы
автоматическое присваивание и поддержка базы данных, 300
база данных, программное создание запросов, 301
уровень абстракции DB, 181, 277, 280
уровни приоритета для различных отладочных комментариев, 230
утилита gzip, сжатие веб-вывода, 220
учетные записи, активация/деактивация для веб-сайта, 240, 242

Ф

фатальные ошибки
E_USER_ERROR, 299
вызванные функциями с одинаковыми именами, 160
файлы, 507, 546
(см. также каталоги), 548
Cache/DB.php, 310
cookies, запись в, 507
i-узлы (inodes), 547
php.ini
session.save_path, параметр, 206
site-search.php, программа, 568, 571
блокировка, 540, 543
базы данных DBM, 284
консультативная блокировка, 541
монопольная, 541
отмена блокировок файлов, 542
текстовые файлы против баз данных, 280
файл как индикатор блокировки, 542
временные, создание, 513
выбор случайной строки из, 522
вывод DOM XML-документа в, 357
загруженные, обработка, 263
запись в несколько файловых дескрипторов одновременно, 534
конфигурационные, чтение, 525, 527
копирование или перемещение, 556
нахождение всех строк, соответствующих шаблону, 401, 404
непосредственная модификация файла без временной копии, 531
обработка всех в каталоге, 557, 559, 561
обработка каждого слова в, 519
обработка текстовых полей переменной длины, 524
обработка по строкам или абзацам в обратном направлении, 522
подсчет строк, абзацев или записей в, 517
получение информации о, 551
права доступа (см. права доступа), 549
преобразование метасимволов среды в, 535
рандомизация всех строк в, 523
сброс вывода в, 532
сжатые
извлечение из ZIP-архива, 545
чтение и запись, 543
символические ссылки, 548
создание или открытие локальных, 511
удаление, 556
удаление последней строки из, 529, 530
удаленные, открытие, 514
хранение зашифрованных данных в, 424
получение файла и дешифрование данных, 425
чтение XML с помощью DOM, 357
чтение в строку, 515
чтение из/запись с определенного места в, 528

чтение данных из, 510
чтение определенной строки в, 521
фигурные скобки (см. {}, в разделе «Символы»), 365
форма регистрации для аутентификации, основанной на cookie, 216, 217
формат файлов PNG, 432
форматирование
даты и времени, 72
строки времени, 71
числа, 57
sprintf, 63
печать шестнадцатеричных чисел, 64
форматы денежного обращения для локалей, 459, 467
формы, 272
безопасная обработка, 265
загруженные файлы, обработка, 263
многократное представление, предотвращение, 261
многостраничные, работа с, 255, 258
обработка ввода, 251
обработка удаленных переменных с точками в именах, 268
повторный вывод форм с ранее сохраненной информацией и сообщениями об ошибках, 258
проверка корректности ввода, 253
проверка с помощью хеширования в скрытых полях, 411
создание выпадающих меню на основе текущей даты, 271
элементы с несколькими значениями, 270
французский республиканский календарь, 96
преобразование даты в юлианское представление и обратно, 97
функции, 160, 177
DB, 285
DOM, 358
ereg (расширенные регулярные выражения), 388
переход к preg, 391, 393
превращение метасимволов в escape-последовательности, 404
FTP, 495
pc_ в именах, 20
PHP, связь с методами XML-RPC, 373, 374
preg (Perl-совместимые регулярные выражения), 388
переход от функций ereg к, 391, 393
ограничители шаблона, 392
превращение метасимволов в escape-последовательности, 404
switch, оператор, внутри, 151
анализ XML, 361
включение встроки, 34
возвращаемые значения, пропуск определенных, 171
возвращение более одного значения, 170, 171
возвращение значений по ссылке, 169
возвращение локализованных сообщений, 464
возвращение ошибки из, 173
вспомогательные (PHP), для XML-RPC, 369
гиперболические, 60
глобальные переменные, доступ внутри, 175
динамические, создание, 177
значение времени выполнения каждой итерации, 238
информация о файле, 548
каталоги, работа с, 547
конструктор класса, 183
оболочка, для отладочной информации, 230
обработка ошибок, 224
обработка элементов массива, передача элементов, 109
обратного вызова
запуск при возникновении ошибки базы данных, 298
обработка сигналов с помощью, 590
меню в окнах GTK, 593
объявление, 160
параметры
доступ, 161
именованные, 165
передача по ссылке, 164
переменное количество, 167, 169
установка значений по умолчанию, 162

переменная, вызов, 174
почта, отправка, 480
преобразование календаря, 97
принимающие базовые объекты, 195
расширение XML-RPC, 369
рисование дуг и эллипсов, 439
тригонометрические, 59
файлы, сжатые gzip, 543
шифрование, 407
функции-оболочки, дополнительная отладочная информация в, 230
функции-обработчики, 189, 190
обработка ошибок, 225
функциональность магических кавычек, обратное преобразование данных и, 155

Х

хеши
(*см. также* ассоциативные массивы), 101
проверка данных с помощью, 410
хосты, пингование, 503
хранилище адресов (LDAP), 498

Ц

цвет фона, установка, 435
цвета
ImageColorAllocate(), функция, 435
веб-корректные, коды для, 64
графическая гистограмма, 456
дуги, эллипсыи окружности, определение, 439
линии, прямоугольники и многоугольники, определение, 437
прозрачные, 451
текст, нарисованный в графическом виде, PostScript, шрифты, 443
фигуры, нарисованные узорными линиями, 440
чередование белых и черных пикселов, 440
цветовые комбинации RGB, 435
определение для графических гистограмм, 456
целые, 47
0 (нуль), как пустая переменная, 145
в левой части сравнения, 147
инициализация массива диапазоном, 106
округление чисел с плавающей точкой до целого, 50
преобразование в числа с плавающей точкой, 47
работа с диапазоном, 51
центрированный текст, рисование на изображении, 444, 449
циклы, 51
for (см. for, циклы), 498
foreach (см. foreach, циклы)
выход с помощью break, 123
изменение порядка элементов массива на обратный, 124
итерация по массивам, 107
по каждому дню месяца, 96
соединение массивов, 117

Ч

часовые пояса
определение времени в различных, 88, 93
смещения между часовыми поясами, 89, 93
числа, 47, 64
логарифмы, 55
начинающиеся с нуля PHP трактует как, 87
основания, отличные от десятичного, 63
порядки, 56
правильная печать во множественном числе, 57
преобразование в строки и обратно, 47
преобразование между системами счисления, 62
проверка на принадлежность к типу, 49
проверка правильности записи числа в строках, 48
с плавающей точкой, 47
выходящие из дапазона допустимых в PHP значений, 61
округление, 50
сравнение, 49
точность, 47
случайные
источники для векторов инициализации, 422

сгенерированные в пределах диапазона, 52
со смещением, генерация, 54
тригонометрические функции, 59
градусы, использование, 60
форматирование, 57
форматирование для различных локалей, 459
целые (см. целые), 51
числовые массивы
режим выборки DB, 288
смешивание и сопоставление со строковыми ключами, 105
соединение, двойные значения и, 135
сортировка, 125
удаление двойных элементов, 133
числовые строки, 48
чтение
из стандартного потока, 515
с определенного места в файле, 528
файла в строку, 515

Ш

шаблоны
Smarty, 339
показ строк, извлеченных из базы данных, 339
изображений для кнопок, 449
разделение записей, сравнение с регулярными выражениями, 405
сравнение с именами файлов, 558
шестнадцатеричные значения
escape-последовательности в строках в двойных кавычках, 24
преобразование ASCII-кодов символов, 393
шифрование, 407, 431
mcrypt, библиотека, 408
mhash, модуль, алгоритмы хеширования, 412
SSL, 428
одностороннее, 407
потерянные пароли, работа с, 416
проверка данных с помощью хеширования, 410
проверка надежности пароля, 414, 416
с открытым ключом, 430
совместное использование зашифрованных данных с другим веб-сайтом, 426, 428
сокрытие данных с помощью кодирования, 410
хранение паролей, 412, 409
электронная почта, шифрование с помощью GPG, 429
шифрование/дешифрование данных, алгоритмы для, 417, 423
перечень алгоритмов библиотеки mcrypt, 418
шрифты
PostScript Type 1, 441
рисование центрированного текста, 445, 446
TrueType, 441
рисование центрированного текста, 445, 448
встроенные в GD, 441
рисование центрированного текста, 444, 447

Э

экземпляр класса, создание, 178
электронная почта
основанный на веб клиент для, 483
отправка, 478
текстовые адреса, преобразование в гиперссылки mailto:, 387
шифрование с помощью GPG, 429
элементы
HTML, с несколькими значениями (в формах), 212, 270, 364
кодирование URL и, 212
кодирование данных, введенных пользователем, 268
кодирование экземпляра RSS в виде HTML, 364
XML
поиск в дереве DOM, 359
создание нового для документа, 355
узлы, 355
массива (см. массивы), 118
элементы управления окном, 575
выровненные по вертикали, 588
выровненные по горизонтали, 588
показ в окне, 586
нескольких, 587, 590

реакция на действия пользователя, 590
эллипсы, рисование, 438
закрашенных, 439

Ю

Юлианский календарь, 96
юлианское представление дат
обзор системы, 80
преобразование в метки времени и обратно, 88
преобразование в негригорианские календари и обратно, 97
разность между двумя датами, 79

Я

язык C
C-библиотека времени выполнения от Microsoft (msvcrt.dll), 584
strftime(), функция, применение в PHP, 76
языки разметки
HTML (см. HTML), 349
XML (см. XML), 349
языки, хранящиеся в объектах, 472

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-059-6, название «PHP. Сборник рецептов» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.