helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

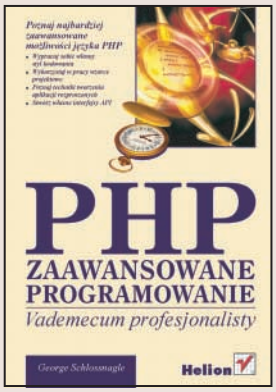
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

PHP. Zaawansowane programowanie. Vademecum profesjonalisty



Autor: George Schlossnagle
Tłumaczenie: Włodzimierz Gajda, Paweł Gonera, Radosław Meryk, Jacek Smycz
ISBN: 83-7361-589-X
Tytuł oryginału: Advanced PHP Programming
Format: B5, stron: 624

PHP to obecnie jeden z najpopularniejszych języków programowania służących do tworzenia aplikacji internetowych. Od początku jego istnienia wykorzystywano go do tworzenia dynamicznych witryn WWW. Dynamiczny rozwój sprawił, że możliwości jego zastosowania znacznie się poszerzyły. Obecnie używa się PHP do tworzenia aplikacji, które jeszcze niedawno wydawały się niemożliwe do napisania w tym języku. Początkowo wykorzystywany przez grono entuzjastów, z czasem stał się stabilną i dobrze udokumentowaną platformą programistyczną, liczącą się na rynku.

Książka „PHP. Zaawansowane programowanie. Vademecum profesjonalisty" to pozycja dla tych, którzy chcą zostać ekspertami w dziedzinie tworzenia aplikacji w PHP. Opisuje najbardziej zaawansowane mechanizmy języka, pozwalające na zastosowanie go w złożonych aplikacjach. Przedstawia techniki programowania obiektowego, testowania jednostek leksykalnych, zagadnienia bezpieczeństwa, techniki buforowania oraz sposoby tworzenia skalowalnych aplikacji internetowych. W książce omówiono również dostrajanie wydajności oraz tworzenie rozszerzeń PHP. Wszystkie, nawet najbardziej skomplikowane kwestie zostały opisane w przejrzysty sposób i zilustrowane kompletnymi przykładami zastosowania w prawdziwych aplikacjach.

- Styl kodowania
- Programowanie obiektowe i wzorce projektowe
- Obsługa błędów
- Korzystanie z szablonów Smarty
- Testowanie modułów
- Zarządzanie wersjami — system CVS
- Techniki buforowania
- Dostęp do baz danych
- Autoryzacja użytkowników
- Tworzenie środowiska rozproszonego
- Wykorzystanie usług sieciowych
- Wydajność aplikacji
- Tworzenie rozszerzeń PHP

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

O Autorze **13**

Słowo wstępne **15**

Wstęp **17**

Część I Implementacja i metodologie programowania **23**

Rozdział 1. Styl kodowania **25**

Wybór wygodnego stylu programowania 26
Formatowanie kodu i układ 26
Wcięcia 26
Długość wiersza 28
Zastosowanie odstępów 29
Zalecenia na temat SQL 29
Instrukcje sterujące 30
Nazewnictwo symboli 35
Zmienne faktycznie globalne oraz stałe 36
Zmienne długowieczne 38
Zmienne tymczasowe 38
Nazwy wieloczłonowe 39
Nazwy funkcji 39
Nazwy klas 40
Nazwy metod 40
Spójność nazw 40
Dopasowanie nazw zmiennych do nazw schematu 40
Unikanie mylącego kodu 41
Unikanie korzystania z otwartych znaczników 42
Unikanie korzystania z funkcji echo do tworzenia kodu HTML 42
Rozsądne korzystanie z nawiasów 43
Dokumentacja 43
Dokumentacja wewnątrz kodu 44
Dokumentacja API 45
Dalsze lektury 49

Rozdział 2. Programowanie obiektowe poprzez wzorce projektowe **51**

Wstęp do programowania obiektowego 52
Dziedziczenie 53
Hermetyzacja 54
Atrybuty i metody statyczne (klasowe) 55
Metody specjalne 56

Krótkie wprowadzenie do wzorców projektowych ... 57
Wzorzec Adaptor ... 58
Wzorzec Template ... 62
Polimorfizm ... 63
Interfejsy i podpowiadanie typów ... 65
Wzorzec Factory ... 67
Wzorzec Singleton ... 68
Przeciążanie ... 70
SPL i iteratory ... 75
__call () ... 80
__autoload() ... 82
Dalsze lektury ... 83

Rozdział 3. Obsługa błędów ... 85
Obsługa błędów ... 87
Wyświetlanie błędów ... 88
Rejestrowanie błędów ... 89
Ignorowanie błędów ... 90
Reagowanie na błędy ... 90
Obsługa błędów zewnętrznych ... 92
Wyjątki ... 94
Wykorzystanie hierarchii wyjątków ... 97
Przykład zastosowania wyjątków ... 99
Wyjątki kaskadowe ... 104
Obsługa błędu działania konstruktora ... 107
Instalowanie wysokopoziomowej procedury obsługi wyjątków ... 108
Kontrola poprawności danych ... 110
Kiedy korzystać z wyjątków? ... 114
Dalsze lektury ... 114

Rozdział 4. Implementacja w PHP — szablony i sieć WWW ... 115
Smarty ... 116
Instalowanie Smarty ... 116
Pierwszy szablon Smarty ... 118
Pod maską skompilowanych szablonów ... 118
Struktury sterujące Smarty ... 119
Funkcje Smarty i więcej ... 122
Buforowanie w Smarty ... 124
Zaawansowane funkcje Smarty ... 126
Tworzenie własnego systemu szablonów ... 127
Dalsze lektury ... 128

Rozdział 5. Implementacja w PHP — samodzielne skrypty ... 131
Wprowadzenie do interfejsu wiersza poleceń PHP (CLI) ... 133
Obsługa wejścia-wyjścia ... 133
Analizowanie argumentów wiersza poleceń ... 136
Tworzenie procesów potomnych i zarządzanie nimi ... 138
Zamykanie współdzielonych zasobów ... 139
Współdzielenie zmiennych ... 139
Porządkowanie po procesach potomnych ... 139
Sygnały ... 141

Tworzenie demonów ... 146
Zmiana katalogu roboczego ... 147
Zmniejszanie uprawnień ... 147
Gwarantowanie wyłączności ... 148
Łączymy wszystko razem — kontrolowanie usług ... 148
Dalsze lektury ... 157

Rozdział 6. Testowanie modułów ... 159
Wprowadzenie do testowania modułów ... 160
Tworzenie testów dla automatycznego testowania modułów ... 161
Pierwszy test modułu ... 161
Dodawanie wielu testów ... 162
Tworzenie wewnętrznych i zewnętrznych testów modułów ... 163
Testy wewnątrz modułu ... 164
Testy na zewnątrz modułu ... 165
Jednoczesne uruchamianie wielu testów ... 166
Dodatkowe funkcje w PHPUnit ... 168
Tworzenie opisowych komunikatów błędów ... 168
Dodawanie większej liczby warunków testowych ... 169
Zastosowanie metod setUp() oraz tearDown() ... 171
Dodawanie nasłuchów ... 171
Wykorzystanie interfejsu graficznego ... 173
Projektowanie sterowane testami ... 173
Kalkulator oceny Flescha ... 174
Testowanie klasy Word ... 175
Raport błędów ... 182
Testowanie modułów w środowisku WWW ... 184
Dalsze lektury ... 186

Rozdział 7. Zarządzanie środowiskiem programistycznym ... 187
Kontrola wersji ... 188
Podstawy CVS ... 189
Modyfikacja plików ... 192
Badanie różnic między plikami ... 193
Mechanizmy usprawniające pracę wielu programistów na tym samym projekcie ... 195
Znaczniki symboliczne ... 197
Gałęzie ... 198
Zarządzanie środowiskiem produkcyjnym oraz programistycznym ... 199
Zarządzanie pakietami ... 203
Tworzenie pakietów i przenoszenie kodu ... 204
Tworzenie pakietów plików binarnych ... 207
Tworzenie pakietu Apache ... 208
Tworzenie pakietu PHP ... 209
Dalsze lektury ... 209

Rozdział 8. Projektowanie dobrego API ... 211
Projektowanie zapewniające możliwości rozszerzania i łatwość modyfikacji ... 212
Umieszczenie logiki w funkcjach ... 212
Tworzenie prostych klas i funkcji ... 214
Przestrzenie nazw ... 214
Zmniejszanie sprzężeń ... 216
Kodowanie defensywne ... 217
Wprowadzenie standardowych konwencji ... 218
Użycie technik oczyszczania danych ... 218
Dalsze lektury ... 220

Część II Buforowanie 221

Rozdział 9. Strojenie wydajności zewnętrznej 223
Techniki dostrajania na poziomie języka 223
Bufory kompilatora 223
Optymalizatory 226
Akceleratory HTTP 227
Odwrotne serwery proxy 228
Dostrajanie systemu operacyjnego 233
Bufory proxy 234
Aplikacje PHP przystosowane do wykorzystania buforowania 234
Kompresja 239
Dalsze lektury 240
Dokumenty RFC 240
Bufory kompilatora 240
Bufory proxy 240
Kompresja 241

Rozdział 10. Buforowanie komponentów danych 243
Właściwości buforowania 243
Identyfikacja komponentów danych, które nadają się do buforowania 245
Wybór właściwej strategii: klasy własne czy biblioteczne? 245
Buforowanie wyników 246
Buforowanie w pamięci 248
Bufor w postaci pojedynczego pliku 248
Zarządzanie rozmiarem bufora 249
Współbieżny dostęp i spójność buforów 250
Buforowanie z wykorzystaniem plików DBM 256
Współbieżny dostęp i spójność buforów 257
Weryfikacja bufora i zarządzanie 257
Buforowanie we współdzielonej pamięci 262
Buforowanie z wykorzystaniem plików cookie 263
Zarządzanie rozmiarem bufora 267
Współbieżny dostęp i spójność buforów 268
Integracja buforowania w kodzie aplikacji 268
Buforowanie stron głównych 270
Wykorzystanie modułu mod_rewrite serwera Apache w celu wykonania inteligentnego buforowania 276
Buforowanie części stron 280
Implementacja bufora zapytań 282
Dalsze lektury 284

Rozdział 11. Ponowne wykorzystanie obliczeń 285
Wstęp z przykładem — ciągi Fibonacciego 285
Buforowanie danych do wielokrotnego użytku wewnątrz żądań 291
Buforowanie wielokrotnie wykorzystywanych danych pomiędzy żądaniami 293
Wielokrotne wykorzystywanie obliczeń w PHP 297
Wyrażenia PCRE 297
Liczniki rozmiarów tablic 297
Dalsze lektury 298

Część III Aplikacje rozproszone 299

Rozdział 12. Współpraca z bazami danych 301
Działanie baz danych i zapytań 302
Analiza zapytań przy użyciu EXPLAIN 305
Wyznaczanie zapytań wymagających profilowania 306
Wzorce dostępu do bazy danych 308
Zapytania ad hoc 309
Metoda aktywnego rekordu 309
Metoda odwzorowania 312
Metoda zintegrowanego odwzorowania 317
Dostrajanie dostępu do bazy danych 318
Ograniczanie zbioru wyników 318
Opóźniona inicjalizacja 320
Dalsze lektury 323

Rozdział 13. Autoryzacja użytkowników oraz bezpieczeństwo sesji 325
Proste schematy uwierzytelniania 326
Proste uwierzytelnianie HTTP 327
Przesyłanie danych identyfikacyjnych w zapytaniu 327
Pliki cookie 328
Rejestrowanie użytkowników 329
Ochrona haseł 329
Zabezpieczenie haseł przed atakami socjotechnicznymi 332
Utrzymanie uwierzytelniania: sprawdzenie, czy w dalszym ciągu komunikujemy się z tą samą osobą 333
Sprawdzenie, czy wartość $_SERVER ['REMOTE_IP'] pozostaje taka sama 333
Sprawdzenie, czy wartość $_SERVER ['USER_AGENT'] pozostaje taka sama 334
Wykorzystanie niezaszyfrowanych plików cookie 334
Zalecane rozwiązania 334
Przykładowa implementacja uwierzytelniania 336
Pojedyncze logowanie 341
Implementacja techniki pojedynczego logowania 343
Dalsze lektury 348

Rozdział 14. Obsługa sesji 349
Przechowywanie danych sesyjnych po stronie klienta 350
Implementacja sesji przy użyciu plików cookie 351
Nieco lepsza pułapka na myszy 353
Przechowywanie danych sesyjnych po stronie serwera 354
Śledzenie identyfikatora sesji 356
Krótkie wprowadzenie do sesji w PHP 358
Niestandardowe funkcje obsługi sesji 360
Usuwanie niepotrzebnych danych 365
Przechowywanie stanu sesji po stronie serwera oraz po stronie klienta 366

Rozdział 15. Tworzenie środowiska rozproszonego 369
Czym jest klaster? 369
Projektowanie klastrów — podstawy 372
Planowanie niepowodzeń 373
Zgodna współpraca 373
Dystrybucja zawartości w klastrze 375
Skalowanie poziome 376
Klastry wyspecjalizowane 377

Buforowanie w środowisku rozproszonym ... 377
Bufory scentralizowane ... 380
W pełni zdecentralizowany bufor wykorzystujący Spread ... 382
Skalowanie baz danych ... 385
Tworzenie aplikacji korzystających z konfiguracji główny-podległy ... 388
Alternatywy replikacji ... 390
Alternatywy systemów zarządzania relacyjnymi bazami danych ... 391
Dalsze lektury ... 392

Rozdział 16. RPC — współpraca ze zdalnymi usługami ... 393
XML-RPC ... 394
Tworzenie serwera: implementacja interfejsu MetaWeblog ... 396
Automatyczne wykrywanie usług XML-RPC ... 400
SOAP ... 402
WSDL ... 404
Zamiana system.load w usługę SOAP ... 406
Usługi WWW Amazon i typy złożone ... 409
Generowanie kodu proxy ... 411
Porównanie SOAP i XML-RPC ... 411
Dalsze lektury ... 412
SOAP ... 412
XML-RPC ... 412
Dzienniki WWW ... 413
Dostępne publicznie usługi WWW ... 413

Część IV Wydajność ... 415

Rozdział 17. Testy wydajności aplikacji — testowanie całości aplikacji ... 417
Pasywna identyfikacja wąskich gardeł ... 418
Generatory obciążenia ... 420
ab ... 420
httperf ... 421
Daiquiri ... 424
Dalsze lektury ... 424

Rozdział 18. Profilowanie ... 425
Co jest potrzebne w programie profilującym PHP? ... 426
Programy profilujące ... 426
Instalacja i zastosowanie APD ... 427
Przykład śledzenia ... 429
Profilowanie większej aplikacji ... 431
Identyfikacja ogólnego braku efektywności ... 436
Usuwanie zbytecznej funkcjonalności ... 438
Dalsze lektury ... 443

Rozdział 19. Syntetyczne testy wydajności — sprawdzanie bloków kodu i funkcji ... 445
Podstawy testowania wydajności ... 446
Tworzenie szablonu testów wydajności ... 447
Zestaw testów wydajności PEAR ... 448
Tworzenie funkcji testujących ... 450
Dodawanie randomizacji danych do wszystkich iteracji ... 451
Usuwanie kosztów funkcji testujących ... 452
Dodawanie własnych informacji czasowych ... 453
Tworzenie testów bezpośrednich ... 457

Przykłady testów wydajności ... 457
Dopasowanie znaków na początku ciągu ... 458
Rozszerzenia makr ... 459
Interpolacja i łączenie ... 464

Część V Możliwości rozszerzania ... 467

Rozdział 20. Działanie maszyny Zend i PHP ... 469
Jak działa maszyna Zend: kody i tablice operacji ... 470
Zmienne ... 475
Funkcje ... 479
Klasy ... 480
Mechanizmy obsługi obiektów ... 482
Tworzenie obiektów ... 483
Inne ważne struktury ... 483
Cykl życiowy żądania PHP ... 485
Warstwa SAPI ... 486
Jądro PHP ... 488
Interfejs rozszerzeń PHP ... 489
Interfejs Zend Extension ... 490
Jak pasują do siebie wszystkie części? ... 492
Dalsze lektury ... 492

Rozdział 21. Rozszerzanie PHP — część I ... 495
Podstawy rozszerzania ... 496
Tworzenie zrębu rozszerzenia ... 496
Budowa i włączanie rozszerzenia ... 499
Wykorzystanie funkcji ... 500
Zarządzanie typami i pamięcią ... 502
Analiza ciągów ... 505
Manipulacja typami ... 506
Testowanie typów, konwersje i mechanizmy dostępu ... 511
Korzystanie z zasobów ... 514
Zwracanie błędów ... 519
Wykorzystanie punktów zaczepienia modułów ... 519
Przykład: osłona klienta Spread ... 527
MINIT ... 529
MSHUTDOWN ... 529
Funkcje modułu ... 529
Wykorzystanie modułu Spread ... 535
Dalsze lektury ... 536

Rozdział 22. Rozszerzanie PHP — część II ... 537
Implementacja klas ... 537
Tworzenie nowej klasy ... 538
Definiowanie właściwości klasy ... 539
Dziedziczenie ... 541
Definiowanie metod klasy ... 542
Definiowanie konstruktorów klas ... 544
Zgłaszanie wyjątków ... 545
Wykorzystanie obiektów zdefiniowanych przez użytkownika oraz prywatnych zmiennych ... 546
Wykorzystanie metod factory ... 549
Tworzenie i implementacja interfejsów ... 550

Pisanie własnych procedur obsługi sesji ... 551
Interfejs API obsługi strumieni ... 555
Dalsze lektury ... 565

Rozdział 23. Tworzenie modułów SAPI oraz rozszerzanie maszyny Zend ... 567
Interfejsy SAPI ... 567
CGI SAPI ... 568
Interfejs Embed SAPI ... 577
Filtry wejściowe interfejsu SAPI ... 578
Modyfikacja i introspekcja mechanizmu maszyny Zend ... 583
Ostrzeżenia jako wyjątki ... 584
Program przetwarzający kod operacyjny ... 586
APD ... 589
APC ... 590
Wykorzystanie wywołań rozszerzenia Zend ... 590
Praca domowa ... 593

Dodatki ... 595

Skorowidz ... 597

Rozdział 9.

Strojenie wydajności zewnętrznej

W każdym przypadku dostrajania nie można utracić obrazu całości systemu. O ile celem cząstkowym bywa przyspieszenie działania określonej funkcji lub strony, głównym celem zawsze jest przyspieszenie działania aplikacji jako całości. Czasami wykonanie zmian w jednym elemencie aplikacji przyczynia się do poprawy jej ogólnej wydajności.

Największe znaczenie dla uzyskania wysokiej wydajności ma uważnie wykonany i solidny projekt oraz zastosowanie właściwych technik programistycznych. Nic tego nie zastąpi. Istnieją jednak techniki dostrajania na zewnątrz kodu PHP, które pozwalają na poprawę wydajności aplikacji. Zastosowanie technik na poziomie serwera lub języka nie zneutralizuje nieudolnego lub nieefektywnego kodowania, ale zapewni, że aplikacja będzie działała najlepiej, jak to możliwe.

W niniejszym rozdziale zamieszczono przegląd kilku technik i produktów, które pozwalają na poprawę wydajności aplikacji. Ponieważ opisane techniki dotyczą wewnętrznych mechanizmów PHP albo też są to produkty zewnętrzne, w niniejszym rozdziale nie ma zbyt wiele kodu. Nie powinno to jednak zniechęcić czytelnika do lektury — czasami największe korzyści uzyskuje się dzięki odpowiedniej kombinacji stosowanych technik.

Techniki dostrajania na poziomie języka

Techniki dostrajania *na poziomie języka* to modyfikacje poprawiające ogólną wydajność aplikacji, które można wykonać w konfiguracji PHP. Język PHP jest wyposażony w wygodny interfejs programowy aplikacji (opisany szczegółowo w rozdziale 21. oraz w rozdziale 23.) pozwalający na pisanie rozszerzeń mających bezpośredni wpływ na działanie mechanizmów języka i sposób wykonywania kodu. Zastosowanie tego interfejsu umożliwia przyspieszenie procesu kompilacji i wykonywania skryptów PHP.

Bufory kompilatora

Gdyby można było wybrać tylko jedną modyfikację konfiguracji serwera, która miałaby przyczynić się do poprawienia wydajności aplikacji PHP, zainstalowanie bufora kompilatora byłoby najwłaściwszym wyborem. Zainstalowanie tego mechanizmu przynosi

olbrzymie korzyści. Dodatkowo, w odróżnieniu od innych technik, których efekty stosowania słabną w miarę zwiększania się rozmiarów aplikacji, w przypadku bufora kompilatora dla bardziej rozbudowanych i złożonych aplikacji efekty okazują się lepsze.

Zatem, czym jest bufor kompilatora i jak to się dzieje, że jego zastosowanie umożliwia uzyskanie tak znacznego wzrostu wydajności? Aby odpowiedzieć na te pytania, spróbujmy przyjrzeć się sposobowi wykonywania skryptów PHP przez maszynę Zend. Kiedy przeglądarka uruchamia skrypt PHP, wykonuje proces składający się z dwóch etapów:

1. Serwer PHP odczytuje plik, analizuje zapisany w nim kod i generuje kod pośredni przeznaczony do wykonywania na wirtualnej maszynie Zend. *Kod pośredni* to termin stosowany w informatyce, opisujący wewnętrzną reprezentację kodu źródłowego skryptu po jego skompilowaniu.
2. Serwer PHP wykonuje kod pośredni.

W procesie tym należy zwrócić uwagę na kilka istotnych elementów:

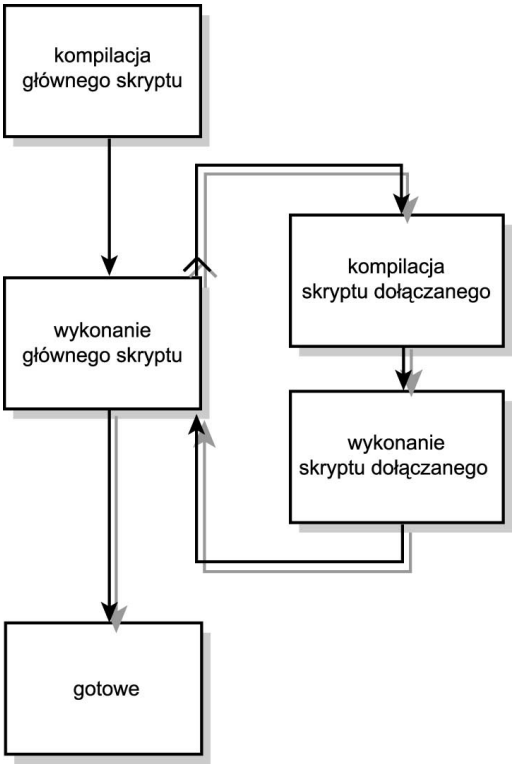
- w przypadku wielu skryptów (szczególnie takich, w których jest dużo instrukcji `include()`) — więcej czasu zajmuje przeanalizowanie skryptu i jego przekształcenie na postać pośrednią niż samo wykonanie kodu;
- nawet jeśli wyniki wykonania etapu 1. niewiele różnią się pomiędzy poszczególnymi wywołaniami skryptu, każde jego wywołanie powoduje wykonywanie opisanych wyżej dwóch etapów;
- sekwencja dwóch etapów wykonywana jest nie tylko wtedy, kiedy wykonywany jest główny skrypt, ale także za każdym razem, kiedy skrypt jest wywoływany za pomocą poleceń `require()`, `include()` lub `eval()`.

Jak łatwo wywnioskować z powyższego opisu, umieszczenie w buforze kodu pośredniego uzyskanego w wyniku wykonania etapu 1. i wykorzystanie go do każdego wywołania skryptu może przynieść duże korzyści. Właśnie na tym polega działanie bufora kompilatora.

Na rysunku 9.1 zilustrowano proces wykonywania skryptu bez wykorzystania bufora kompilatora, natomiast na rysunku 9.2 — z wykorzystaniem bufora. Warto zwrócić uwagę, że tylko pierwsze uruchomienie skryptu lub odwołanie się do niego w instrukcji `include` powoduje konieczność tworzenia kodu pośredniego i jego umieszczania w buforze. W każdym kolejnym przypadku etap kompilacji skryptu jest pomijany całkowicie.

Dla języka PHP istnieją trzy najbardziej popularne bufory kompilatorów:

- **Zend Accelerator** — komercyjny bufor kompilatora bez dostępu do kodu źródłowego, produkowany przez firmę Zend Industries i rozprowadzany za opłatą;
- **ionCube Accelerator** — komercyjny bufor kompilatora bez dostępu do kodu źródłowego, ale darmowy; napisany przez Nicka Lindridge'a i rozprowadzany przez jego firmę ionCube;
- **APC** — darmowy bufor kompilatora typu *open source*, napisany przez Daniela Cowgilla i mnie.



Rysunek 9.1. *Wykonywanie skryptu w PHP*

W rozdziale 23., w którym opisano rozszerzenia PHP i mechanizm maszyny Zend, szczegółowo omówiono także działanie bufora APC.

Bufor kompilatora APC jest dostępny poprzez bibliotekę rozszerzeń *PECL* (skrót od ang. *PEAR Extension Code Library*). Bibliotekę tę instaluje się za pomocą następującego polecenia:

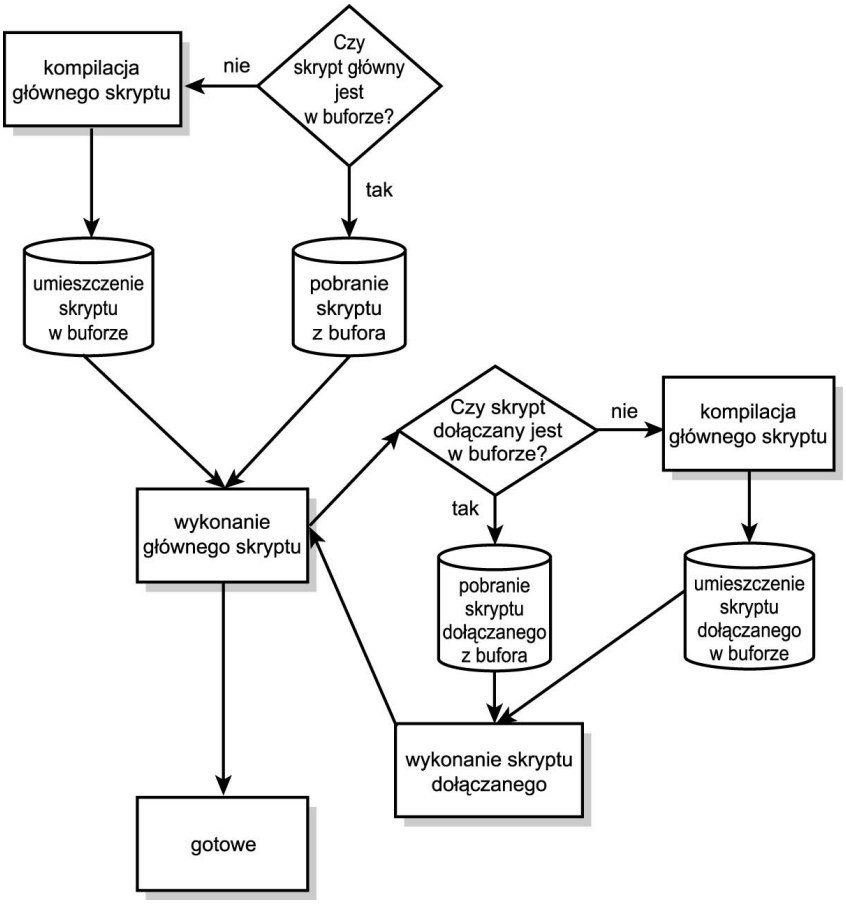
```
#pear install apc
```

Skonfigurowanie bufora wymaga dodania następującego wiersza w pliku *php.ini*:

```
extension = /ścieżka/do/pliku/apc.so
```

Oprócz tego nie trzeba wykonywać żadnych czynności konfiguracyjnych. Przy następnym uruchomieniu serwera PHP bufor APC będzie aktywny (będzie buforował skrypty we współdzielonej pamięci).

Jak pamiętamy, bufor kompilatora umożliwia pominięcie fazy analizy kodu podczas wykonywania skryptu, a zatem najlepsze efekty jego stosowania można uzyskać dla skryptów zawierających dużo kodu. Dla sprawdzenia różnicy porównałem przykładową stronę wykorzystującą szablony, rozprowadzaną z systemem Smarty. W przypadku standardowej konfiguracji PHP udało mi się uzyskać 26 żądań na sekundę. Po załadowaniu bufora APC uzyskałem 42 żądania na sekundę — 61 % poprawy wydajności to znaczący zysk, zwłaszcza, że efekt ten uzyskałem bez modyfikacji kodu.



Rysunek 9.2. *Wykonywanie kodu z zastosowaniem bufora kompilatora*

Bufory kompilatorów pozwalają na uzyskanie największych korzyści w przypadku kodu, w którym występuje duża ilość instrukcji `include()`. Kiedy pracowałem w firmie Community Connect (tam, gdzie powstał bufor APC), zdarzało się, że w skrypcie było, uwzględniając wywołania rekurencyjne, 30, a nawet 40 wywołań tej instrukcji. Taka duża liczba plików dołączanych wynikała z modularnego układu zasadniczej części kodu, w której funkcje o podobnym przeznaczeniu były umieszczane w oddzielnych bibliotekach. W tamtym środowisku zastosowanie bufora APC pozwoliło na uzyskanie ponad stuprocentowego wzrostu wydajności.

Optymalizatory

Działanie optymalizatorów kodu polega na przekształceniu skompilowanego pośredniego kodu skryptu i przeprowadzenie dla niego optymalizacji. W większości języków programowania istnieją optymalizatory wykonujące następujące działania:

- **Eliminacja „martwego" kodu** — usuwanie instrukcji, do których nie ma dostępu, na przykład `if (0){}`.

- **Składanie stałych** — przekształcanie kodu wykonującego operacje na grupie stałych w taki sposób, aby obliczenia były wykonywane raz w fazie kompilacji. Na przykład operację:

  ```
  $seconds_in_day = 24*60*60;
  ```

  optymalizator przekształci wewnętrznie na instrukcję w postaci:

  ```
  $seconds_in_day = 86400;
  ```

  bez konieczności modyfikacji kodu.

- **Optymalizacja wewnętrznej struktury kodu** — lokalne działania optymalizacyjne wykonywane w celu poprawy wydajności kodu (na przykład przekształcenie instrukcji `$count++` na `++$count` w przypadku, kiedy zwracana wartość występuje w kontekście *void*). Instrukcja `$count++` wykonuje inkrementację po obliczeniu wartości wyrażenia `$count`. I tak na przykład wykonanie instrukcji `$i = $count++` powoduje ustawienie zmiennej `$i` na wartość zmiennej `$count` i późniejszą inkrementację zmiennej `$count`. Oznacza to konieczność zapamiętania wartości zmiennej `$count` do wykorzystania w wyrażeniach. W odróżnieniu od tej instrukcji `++$count` powoduje inkrementację przed obliczeniem wartości wyrażenia, a zatem nie trzeba zapamiętywać wartości (dzięki czemu instrukcja wykonuje się szybciej). Jeżeli instrukcja `$count++` jest wykorzystywana w wyrażeniu, w którym wartość zmiennej `$count` nie jest wykorzystywana (co określa się jako *kontekst void*), można ją bezpiecznie przekształcić na instrukcję `++$count`.

Optymalizatory można również wykorzystać do wielu innych zastosowań.

W języku PHP nie ma wewnętrznego optymalizatora kodu, ale istnieją dodatki spełniające takie funkcje:

- optymalizator Zend — bez dostępu do kodu źródłowego, ale darmowy;
- wbudowany optymalizator akceleratora ionCube;
- optymalizator typu *proof-of-concept* w bibliotece PEAR.

Największe korzyści z zastosowania optymalizatorów kodu uzyskuje się w przypadku, kiedy kod jest kompilowany i optymalizowany raz, a potem uruchamiany wielokrotnie. Tak więc w języku PHP korzyści z zastosowania optymalizatora bez bufora kompilatora są minimalne. Optymalizator użyty w połączeniu z buforem kompilatora pozwala na uzyskanie niewielkiej, ale zauważalnej poprawy w porównaniu z użyciem samego bufora kompilatora.

Akceleratory HTTP

Wydajność aplikacji jest zagadnieniem złożonym. Składa się na nią wiele czynników, między innymi:

- wydajność bazy danych;
- wydajność procesora — dla aplikacji, w której są wykonywane intensywne obliczenia;

- wydajność dysku — z uwagi na wykonywanie wielu operacji wejścia-wyjścia (we-wy);
- wydajność sieci — dla aplikacji, w których są przesyłane duże ilości danych sieciowych.

W kilku następnych rozdziałach zostaną opisane sposoby dostrajania aplikacji mające na celu zminimalizowanie efektów tych „wąskich gardeł". Zanim jednak przejdziemy do omawiania tych tematów, zajmijmy się innym „wąskim gardłem", którego się często nie dostrzega, a mianowicie opóźnieniami sieci. Kiedy użytkownik maszyny klienckiej żąda informacji z witryny, pakiety danych muszą być fizycznie przesłane w sieci internet od klienta do serwera i z powrotem. Co więcej — w systemie operacyjnym są ograniczenia ilości danych, jakie można jednorazowo przesyłać przez gniazdo TCP. W przypadku przekroczenia tego limitu aplikacja blokuje transfer danych lub po prostu czeka do chwili potwierdzenia otrzymania danych przez system zdalny. Tak więc oprócz czasu poświęconego na przetwarzanie żądania serwer WWW, który je obsługuje, musi jeszcze oczekiwać przez czas opóźnienia spowodowany wolnym połączeniem sieciowym.

Na rysunku 9.3 zilustrowano wykonywane w sieci działania związane z obsługą pojedynczego żądania wraz z przykładowym czasem ich wykonywania. Podczas przesyłania pakietów pomiędzy serwerem a klientem w sieci aplikacja PHP jest zupełnie bezczynna. Zwróćmy uwagę, że w sytuacji pokazanej na rysunku 9.3 serwer PHP jest bezczynny przez 200 ms i oczekuje na zakończenie transmisji w sieci pomimo tego, że jest gotowy do obsługi żądania. W wielu aplikacjach czas opóźnień sieciowych jest większy od czasu wykonywania skryptów.

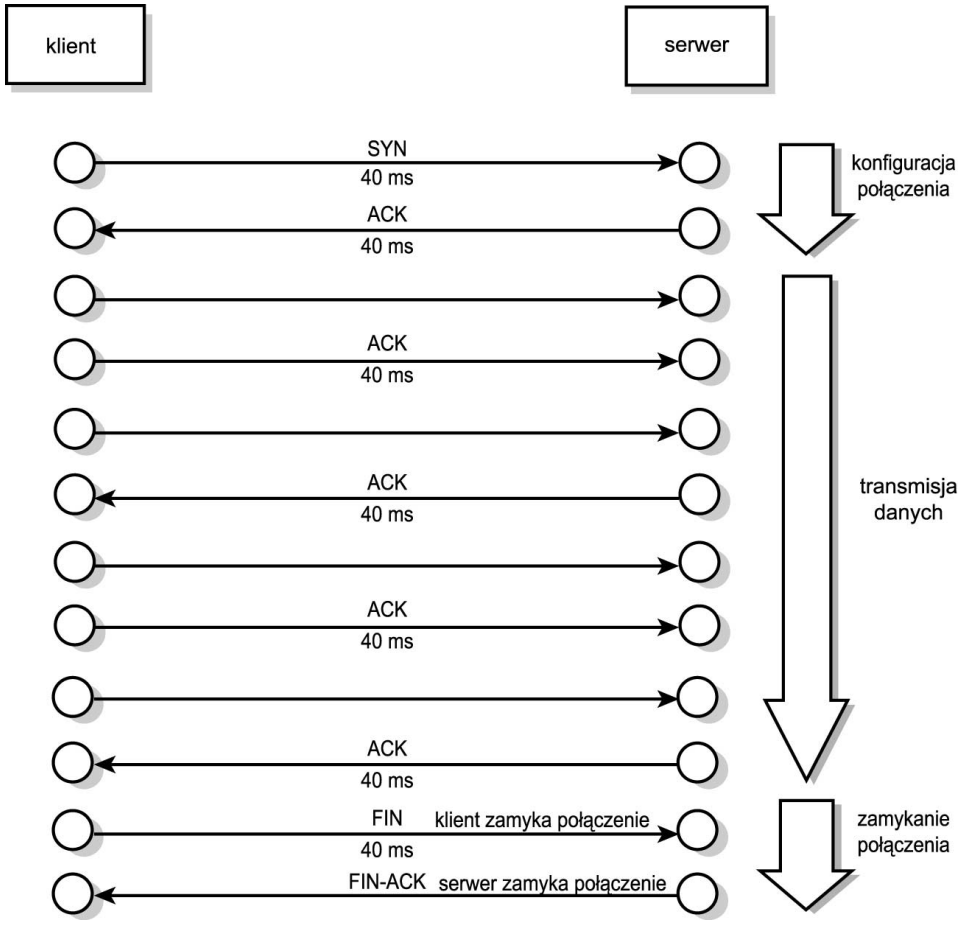
Choć wydaje się, że bezczynność serwera nie jest wąskim gardłem, okazuje się, że również może nim być. Problem polega na tym, że nawet bezczynny serwer WWW zużywa zasoby: pamięć, trwałe połączenia z bazą danych oraz miejsce w tablicy procesów. Jeśli uda się wyeliminować opóźnienia w sieci, można zmniejszyć ilość czasu, w którym procesy serwera PHP wykonują nieistotne działania, i w ten sposób poprawić wydajność.

Blokowanie połączeń sieciowych

Określenie, że aplikacja musi zablokować połączenie sieciowe, nie jest do końca precyzyjne. Gniazda sieciowe można tworzyć w taki sposób, że zamiast blokowania sterowanie wraca do aplikacji. Metoda ta jest stosowana w wielu wysoko wydajnych serwerach WWW, takich, jak thttpd i Tux. Oprócz nich nie są mi znane interfejsy API serwera PHP (SAPI — aplikacje z wbudowanym serwerem PHP), które umożliwiają jednemu serwerowi PHP jednoczesną obsługę wielu żądań. Tak więc pomimo tego, że połączenie sieciowe nie blokuje aplikacji, nawet szybkie serwery wymagają dedykowanego procesu PHP przez cały czas trwania obsługi żądania klienta.

Odwrotne serwery proxy

Niestety, wyeliminowanie opóźnień sieciowych w internecie leży poza zakresem naszych możliwości (a szkoda). Można jednak zainstalować dodatkowy serwer pomiędzy użytkownikiem a aplikacją PHP, który będzie odbierał żądania od klientów, przekazywał kompletne żądania do aplikacji PHP, oczekiwał na odpowiedź, a następnie przesyłał ją do użytkownika zdalnego. Taki „wtrącony" serwer nazywa się *odwrotnym serwerem proxy* (ang. *reverse proxy*), a czasami *akceleratorem HTTP*.

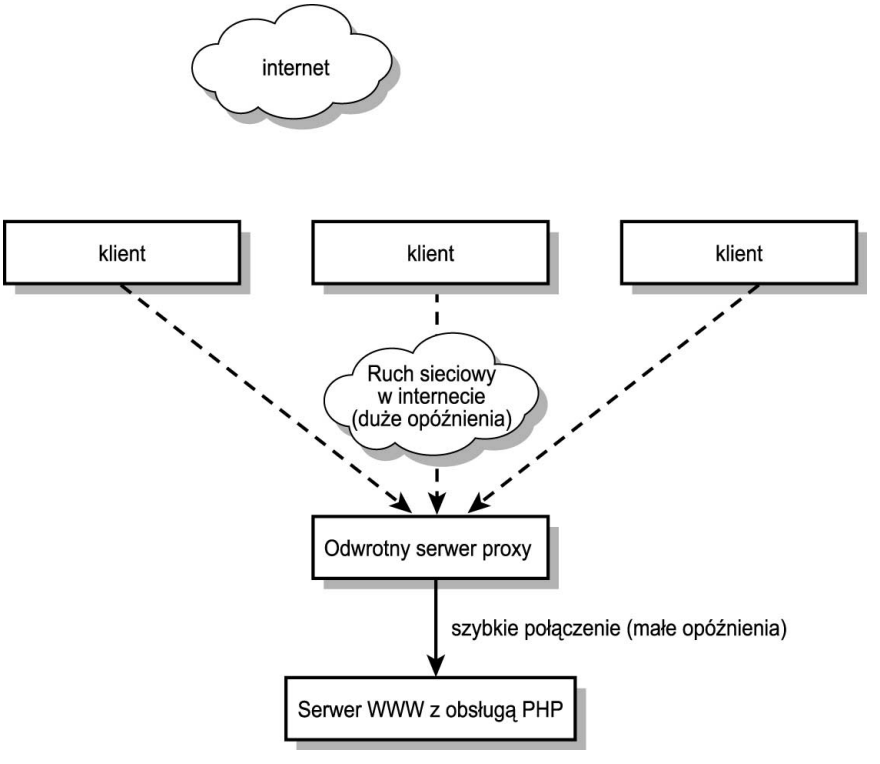


Rysunek 9.3. *Czasy transmisji sieciowej dla typowego żądania*

Zastosowanie takiej strategii wymaga spełnienia kilku warunków:

- Serwer proxy nie może zużywać zbyt wielu zasobów. Dla jednego żądania klienta serwer proxy zużywa znacznie mniej zasobów niż aplikacja PHP.
- Serwer proxy oraz aplikacja PHP muszą znajdować się w tej samej sieci lokalnej. Dzięki temu połączenie pomiędzy aplikacją, a serwerem wprowadza bardzo małe opóźnienie.

Typową konfigurację odwrotnego serwera proxy pokazano na rysunku 9.4. Warto zwrócić uwagę, że zdalne klienty są podłączone do wolnych łącz (z dużymi opóźnieniami), natomiast serwer proxy i serwer WWW znajdują się w tej samej, szybkiej sieci. Zwróćmy również uwagę na to, że serwer proxy podtrzymuje znacznie więcej połączeń klienckich niż połączeń z serwerem WWW. Taka sytuacja jest spowodowana tym, że dzięki szybkiemu łączu pomiędzy serwerem WWW a serwerem proxy serwer WWW może obsługiwać żądania na bieżąco i nie musi marnować czasu w oczekiwaniu na zakończenie transmisji sieciowej.



Rysunek 9.4. *Typowa konfiguracja odwrotnego serwera proxy*

W przypadku serwera Apache mamy do wyboru kilka doskonałych odwrotnych serwerów proxy, między innymi:

- `mod_proxy` — standardowy moduł dostarczany wraz z serwerem Apache;
- `mod_accel` — moduł produkowany przez firmę zewnętrzną, bardzo podobny do modułu *mod_proxy* (w dużej części wykorzystano w nim kod źródłowy modułu *mod_proxy*), oferujący dodatkowo funkcje specyficzne dla odwrotnych serwerów proxy;
- `mod_backhand` — moduł równoważenia obciążenia, produkowany przez firmę zewnętrzną, spełniający funkcję odwrotnego serwera proxy;
- **Squid** — zewnętrzny demon serwera proxy oferujący wysoko wydajne funkcje zwykłego (działającego w trybie *forward*) i odwrotnego serwera proxy.

We wszystkich wymienionych rozwiązaniach egzemplarz serwera proxy może działać na dedykowanym komputerze lub też jako oddzielny egzemplarz serwera na tym samym komputerze. Spróbujmy przeanalizować sposób konfiguracji odwrotnego serwera proxy działającego na tym samym komputerze z wykorzystaniem serwera *mod_proxy*. Najłatwiejszym sposobem wykonania takiej konfiguracji jest zainstalowanie dwóch kopii serwera Apache — jednej z wbudowanym modułem *mod_proxy* (zainstalowanym w katalogu */opt/apache_proxy*) oraz drugiej z serwerem PHP (zainstalowanym w katalogu */opt/apache_php*).

W tym celu wykorzystamy znaną sztuczkę umożliwiającą wykorzystanie tej samej konfiguracji serwera Apache na wielu komputerach: w pliku konfiguracyjnym serwera Apache zdefiniujemy hosta o nazwie `externalether`. Następnie odwzorujemy go na publiczny (zewnętrzny) interfejs ethernetowy w pliku */etc/hosts*. Podobnie zdefiniujemy hosta *localhost* w pliku konfiguracyjnym serwera, który będzie odwzorowywany na adres pętli zwrotnej 127.0.0.1.

Umieszczenie całej konfiguracji serwera Apache w niniejszym rozdziale zajęłoby zbyt dużo miejsca. Zamiast tego w celu pokazania najważniejszych ustawień zaprezentujemy zaledwie niewielki fragment pliku *httpd.conf*.

Konfiguracja odwrotnego serwera proxy z wykorzystaniem modułu *mod_proxy* wymaga wprowadzenia następujących ustawień:

```
DocumentRoot /dev/null
Listen          externalether:80
MaxClients      256
KeepAlive       Off
AddModule mod_proxy.c
ProxyRequests On
ProxyPass        / http://localhost/
ProxyPassReverse / http://localhost/
ProxyIOBufferSize 131072
<Directory proxy :*>
         Order Deny, Allow
         Deny from all
</Directory>
```

W przypadku pokazanej konfiguracji należy zwrócić uwagę na następujące elementy:

- Katalog `DocumentRoot` jest ustawiony na */dev/null*, ponieważ serwer nie posiada własnej zawartości.
- W konfiguracji określono jawne dowiązanie do zewnętrznego adresu ethernetowego serwera (`externalether`). Powiązanie należy określić jawnie, ponieważ w przypadku tej konfiguracji na jednym komputerze działają dwa oddzielne egzemplarze serwera PHP. Bez instrukcji `Listen` pierwszy serwer rozpoczynający działanie wiązałby wszystkie dostępne adresy, uniemożliwiając działanie drugiego egzemplarza.
- Parametr `KeepAlive` ustawiono na `Off`. Serwery WWW obsługujące duży ruch, w których stosuje się model *pre-fork* (np. Apache), lub (w mniejszym stopniu) takie, w których stosuje się modele wielowątkowe (np. Zeus), zazwyczaj działają mniej wydajnie przy ustawieniu parametru `KeepAlive` na `On`.
- Parametr `ProxyRequests` ma wartość `On`, co uaktywnia moduł *mod_proxy*.
- Instrukcja `ProxyPass / http://localhost/` powoduje, że moduł *mod_proxy* wewnętrznie pośredniczy w przekazywaniu żądań rozpoczynających się od znaku / (czyli wszystkich żądań) do serwera powiązanego z adresem IP hosta *localhost* (tzn. z serwerem PHP).

- Jeśli serwer PHP wyśle do skryptu *foo.php* żądanie przekierowania zawierające nazwę serwera, klient otrzyma to żądanie w następującej postaci:

  ```
  Location: http://localhost/foo.php
  ```

  W przypadku użytkownika zdalnego takie żądanie nie zadziała, a zatem zmienna `ProxyPassReverse` przepisuje przeadresowania `Location` w taki sposób, aby wskazywały na właściwego hosta.

- Zapis `ProxyIOBufferSize 131072` powoduje ustawienie rozmiaru bufora wykorzystywanego przez odwrotny serwer proxy do zbierania informacji przekazywanych przez serwer PHP na 131072 bajty. Aby zapobiec blokowaniu serwera podczas komunikacji z przeglądarką, parametr ten należy ustawić na wartość równą co najmniej największemu rozmiarowi strony przesyłanej do użytkownika. Dzięki temu można przesłać całą stronę z serwera PHP do serwera proxy, zanim jakiekolwiek dane zostaną przesłane z powrotem do przeglądarki. W takim przypadku, podczas gdy serwer proxy obsługuje przesyłanie danych do przeglądarki klienckiej, serwer PHP może kontynuować działanie.

- Na zakończenie należy wyłączyć wszystkie żądania proxy wychodzące do serwera. Ustawienie to zapobiega powstawaniu błędów typu *open proxy*.

Architektury procesów: pre-fork, z obsługą zdarzeń oraz wielowątkowa

Serwery WWW mogą działać w trzech różnych architekturach: w architekturze pre-fork, z obsługą zdarzeń i wielowątkowej.

W modelu *pre-fork* istnieje pula procesów, których zadaniem jest obsługa nowych żądań. W przypadku pojawienia się nowego żądania jego obsługę powierza się jednemu z procesów potomnych. Proces potomny zwykle obsługuje więcej niż jedno żądanie. Taki model zastosowano w wersji 1.3 serwera Apache.

W modelu *z obsługą zdarzeń* pojedynczy proces obsługuje żądania w jednym wątku, wykorzystując do szybkiej obsługi wielu żądań nieblokujący (asynchroniczny) mechanizm wejścia-wyjścia. Taka architektura działa doskonale w przypadku obsługi plików statycznych, ale niezbyt dobrze sprawdza się dla żądań dynamicznych (ponieważ jest tu potrzebny oddzielny proces lub wątek do dynamicznej części każdego z żądań). Model z obsługą zdarzeń zastosowano w niewielkim, szybkim serwerze WWW autorstwa Jefa Poskanzera — thttpd.

W modelu *wielowątkowym* pojedynczy proces do obsługi żądań wykorzystuje pulę wątków. Model ten jest bardzo podobny do modelu pre-fork, poza tym, że ze względu na istnienie wielu wątków niektóre zasoby są współdzielone pomiędzy wątkami. Taki model wykorzystano w serwerze WWW Zeus. Pomimo tego, że samo PHP obsługuje synchronizację wątków, jest bardzo trudno stwierdzić lub też zagwarantować, że biblioteki firm zewnętrznych wykorzystywane w kodzie rozszerzeń także ją obsługują. Oznacza to, że nawet w przypadku wielowątkowego serwera WWW często nie można zastosować wielowątkowego kodu PHP, ale trzeba zastosować równoległe wykonywanie procesów poprzez implementacje *fastcgi* lub *cgi*.

W serwerze Apache 2 zastosowano architekturę umożliwiającą skonfigurowanie architektury procesów jako pre-fork, wielowątkowej lub hybrydowej, w zależności od potrzeb.

W odróżnieniu od nakładów pracy konfiguracyjnej w obrębie serwera Apache proxy, konfiguracja Apache PHP jest bardzo podobna do standardowej. Jedyną zmianą jest dodanie w pliku *httpd.conf* następującego wiersza:

```
Listen localhost:80
```

Zapis ten powoduje wyłączne dowiązanie egzemplarza serwera Apache PHP do adresu pętli zwrotnej. Po wprowadzeniu tego ustawienia dostęp do serwera WWW wymaga komunikacji za pośrednictwem serwera proxy.

Pomiar efektów wykonanych modyfikacji jest bardzo trudny. Ponieważ zmniejszają one ilość potrzebnych operacji głównie w przypadku obsługi klientów poprzez łącza wprowadzające duże opóźnienia, jest bardzo trudno zmierzyć ich efekty w sieciach lokalnych i bardzo szybkich. W rzeczywistej konfiguracji zauważyłem, że zastosowanie konfiguracji odwrotnego serwera proxy pozwala na zmniejszenie liczby potomnych procesów serwera Apache wymaganych do obsługi witryny ze 100 do 20.

Dostrajanie systemu operacyjnego

Wśród informatyków wciąż toczy się zażarta dyskusja, z której wynika, że jeżeli nie zastosuje się lokalnego buforowania, wykorzystanie konfiguracji odwrotnego serwera proxy wiąże się ze zbyt dużym nakładem w porównaniu z uzyskanymi efektami. Podobny efekt do zastosowania odwrotnego serwera proxy bez konieczności instalowania osobnego serwera można uzyskać, powierzając zadanie buforowania danych systemowi operacyjnemu. Z opisu konfiguracji odwrotnych serwerów proxy we wcześniejszej części niniejszego rozdziału wynika, że największy udział w czasie oczekiwania na przesłanie danych ma czas blokowania klienta pomiędzy przesyłanymi do niego pakietami.

Aplikacja musi wysyłać wiele pakietów, ponieważ w systemie operacyjnym istnieje ograniczenie ilości informacji, jakie można zbuforować przed ich przesłaniem przez gniazdo TCP. Na szczęście to ustawienie można łatwo dostroić.

W systemie FreeBSD konfigurację buforów TCP można zmodyfikować za pomocą następujących poleceń:

```
#sysctl -w net.inet.tcp.sendspace=131072
#sysctl -w net.inet.tcp.recvspace=8192
```

W systemie Linux należy wprowadzić następujące polecenie:

```
#echo "131072" > /proc/sys/net/core/wmem_max
```

Po wykonaniu tych modyfikacji rozmiar bufora wychodzących żądań TCP będzie wynosił 128 kB, natomiast rozmiar bufora żądań przychodzących — 8 kB (ponieważ do klienta dochodzi niewielka ilość danych przychodzących, natomiast odsyłanych jest wiele danych). Przy takim założeniu maksymalny rozmiar wysyłanej strony wynosi 128 kB. Wartość tę należy dostosować do konkretnych realiów. Dodatkowo można dostroić parametr `kern.ipc.nmbclusters` w celu przydzielenia pamięci dla większych buforów (po szczegółowe informacje odsyłam do dokumentacji systemów).

Po dostrojeniu limitów systemu operacyjnego należy odpowiednio skonfigurować serwer Apache, tak aby wykorzystywał zdefiniowane w systemie większe bufory. W tym celu w pliku *httpd.conf* należy wprowadzić następujący wiersz:

```
SendBufferSize 131072
```

Dodatkowo można wyeliminować opóźnienie sieci podczas zamykania połączenia poprzez zainstalowanie w serwerze Apache nakładki *lingerd*. Po zakończeniu połączenia sieciowego wysyłający wysyła odbiorcy pakiet `FIN`, oznaczający zakończenie połączenia. Wysyłający przed zamknięciem gniazda musi zaczekać na otrzymanie od odbierającego potwierdzenia otrzymania pakietu `FIN`. W ten sposób uzyskuje pewność, że wszystkie dane zostały pomyślnie przesłane. Po przesłaniu pakietu `FIN` serwer Apache musi jedynie poczekać na otrzymanie pakietu `FIN-ACK` i zamknąć połączenie. Proces *lingerd* poprawia wydajność tej operacji dzięki przekazaniu gniazda do zewnętrznego demona (*lingerd*), który oczekuje na otrzymanie pakietu `FIN-ACK` po to, aby zamknąć gniazdo.

W przypadku obciążonych serwerów WWW zastosowanie nakładki *lingerd* umożliwia uzyskanie znaczącego zysku wydajności, w szczególności w połączeniu ze zwiększeniem rozmiarów buforów. Kompilacja nakładki *lingerd* jest bardzo prosta. Jest ona wykorzystywana w wielu znanych witrynach WWW, między innymi *Sourceforge.com*, *Slashdot.org* oraz *LiveJournal.com*.

Bufory proxy

Jeszcze lepsze efekty od zastosowania szybkich połączeń do serwera zawartości daje taka konfiguracja, w której w ogóle nie trzeba przesyłać żądań. Takie możliwości uwzględniono w protokole HTTP.

Buforowanie HTTP jest możliwe na wielu poziomach:

- wbudowane buforowanie w odwrotnych serwerach proxy;
- bufory proxy zainstalowane u dostawców internetu;
- bufory wbudowane w przeglądarkach WWW.

Na rysunku 9.5 zaprezentowano typową konfigurację odwrotnego serwera proxy. Jeśli użytkownik przesyła żądanie do strony *www.example.foo*, serwer DNS w rzeczywistości kieruje go do serwera proxy. Jeśli żądana strona istnieje w buforze proxy i nie jest przestarzała, kopia strony z bufora jest przesyłana do użytkownika bez kontaktu z serwerem WWW. W innym przypadku połączenie jest kierowane do serwera WWW w sposób opisany we wcześniejszej części niniejszego rozdziału przy okazji opisywania konfiguracji odwrotnego serwera proxy.

W wielu produktach oferujących funkcje odwrotnego serwera proxy istnieją zintegrowane bufory. Należą do nich *Squid*, *mod_proxy* i *mod_accel*. Wykorzystanie buforowania zintegrowanego z odwrotnym serwerem proxy jest łatwym sposobem usprawnienia działania konfiguracji serwera proxy. Zastosowanie lokalnego buforowania gwarantuje odciążenie serwerów PHP dzięki maksymalnemu wykorzystaniu możliwości buforowania.

Aplikacje PHP przystosowane do wykorzystania buforowania

Aby można było wykorzystać mechanizmy buforowania, aplikacje PHP muszą być do tego odpowiednio przystosowane. W takich aplikacjach należy uwzględnić strategie buforowania przeglądarek i serwerów proxy, a także odpowiednio przygotować dane.

W kodzie aplikacji powinny znajdować się odpowiednie instrukcje wysyłające do przeglądarek dyrektywy sterowania buforami.

Istnieją cztery parametry nagłówka HTTP wykorzystywane do obsługi buforowania w aplikacjach:

- `Last-Modified`
- `Expires`
- `Pragma: no-cache`
- `Cache-Control`

Parametr `Last-Modified` w HTTP 1.0 ma kluczowe znaczenie w negocjacji możliwości buforowania pomiędzy skryptem a przeglądarką. Wartością tego parametru jest data ostatniej modyfikacji strony w formacie *UTC* (skrót od ang. *Universal Time Coordinated*, poprzednio GMT). Kiedy bufor żąda weryfikacji aktualności strony, wysyła datę określoną parametrem `Last-Modified` jako wartość pola `If-Modified-Since`, dzięki czemu serwer wie, z jaką wersją strony ma porównać przesłaną zawartość.

Pole `Expires` jest komponentem protokołu HTTP 1.0 umożliwiającym weryfikację aktualności strony. Jego wartość składa się z daty (w formacie GMT), po której żądana strona traci ważność.

Niektórzy uwzględniają także parametr `Pragma: no-cache`, który oznacza, że dokument nie powinien być buforowany. Chociaż nic nie stoi na przeszkodzie, aby ustawić ten parametr, jego znaczenie określono jawnie jedynie w specyfikacji HTTP 1.0, a zatem jego przydatność wynika głównie z tego, że jest standardem *de facto* zaimplementowanym w wielu buforach obsługujących HTTP 1.0.

W końcu lat dziewięćdziesiątych, kiedy w wielu aplikacjach klienckich wykorzystywano protokół HTTP 1.0, możliwości negocjacji buforowania przez aplikacje były ograniczone. W większości dynamicznych stron standardowo umieszczano następujący nagłówek:

```
function http_1_0_nocache headers()
{
    $pretty_modtime = gmdate('D, d M Y H:i:s'). ' GMT';
        header("Last-Modified: $pretty_modtime");
        header("Expires: $pretty_modtime");
        header("Pragma: no-cache");
}
```

Fragment ten oznacza, że dane nie powinny być umieszczane w buforze i zawsze należy je odświeżać.

Po dokładniejszej analizie możliwości gwarantowanych przez wymienione parametry nagłówkowe można zauważyć istotne niedogodności:

- Ustawienie czasu ważności strony jako bezwzględnego znacznika czasu wymaga synchronizacji zegarów systemowych systemów klienta i serwera.

- Bufor w przeglądarce klienta różni się od bufora u dostawcy usług internetowych. W buforze przeglądarki można zapisywać indywidualne dane użytkownika, natomiast w buforze serwera proxy, który jest wykorzystywany przez wielu użytkowników, nie można tego robić.

Usprawnieniem wymienionych niedogodności zajęto się w specyfikacji protokołu HTTP 1.1, gdzie w celu rozwiązania istniejących problemów dodano zestaw dyrektyw `Cache-Control`. Możliwe wartości parametru `Cache-Control` zdefiniowano w dokumencie RFC 2616. Składnia dyrektywy `Cache-Control` jest następująca:

```
Cache-Control = "Cache-Control" ":" 1#cache-response-directive

cache-response-directive =
  "public"
| "private"
| "no-cache"
| "no-store"
| "no-transform"
| "must-revalidate"
| "proxy-revalidate"
| "max-age" "=" delta-seconds
| "s-maxage" "=" delta-seconds
```

Dyrektywa `Cache-Control` definiuje możliwości buforowania żądanego dokumentu. Zgodnie z dokumentem RFC 2616, dyrektyw `Cache-Control` powinny przestrzegać wszystkie bufory oraz serwery proxy, a nagłówki do przeglądarki wysyłającej żądanie muszą przesyłać wszystkie serwery proxy.

Do określenia możliwości buforowania żądania można wykorzystać następujące dyrektywy:

- `public` — odpowiedź może być umieszczona w dowolnym buforze.
- `private` — odpowiedź może być umieszczona w takim buforze, który nie jest współdzielony. Oznacza to, że żądanie można umieścić wyłącznie w buforze przeglądarki żądającego, natomiast nie można go umieścić w innych buforach.
- `no-cache` — odpowiedzi nie można umieścić w buforze na żadnym z poziomów buforowania. Dyrektywa `no-store` oznacza, że przesyłane informacje są wrażliwe i nie można ich zapisywać na trwałych nośnikach. Dla obiektów, które można buforować, można wprowadzić dyrektywy określające czas, przez jaki obiekty mogą być przechowywane w buforze.
- `must-revalidate` — wszystkie bufory muszą weryfikować żądania strony. W czasie weryfikacji przeglądarka wysyła w żądaniu nagłówek `If-Modified-Since`. Jeśli serwer potwierdzi, że strona jest najnowszą kopią, zwraca do klienta odpowiedź `304 Not Modified`. W innym przypadku powinien przesłać do niego pełną treść strony.
- `proxy-revalidate` — dyrektywa podobna do `must-revalidate`, z tą różnicą, że obowiązek weryfikacji treści dotyczy buforów współdzielonych przez wielu użytkowników.
- `max-age` — czas wyrażony w sekundach, przez który strona może być buforowana bez konieczności weryfikacji.

- `s-maxage` — maksymalny czas, przez jaki dokument zapisany we współdzielonym buforze powinien być uważany za aktualny. Zgodnie ze specyfikacją HTTP 1.1 ustawienie dyrektyw `max-age` lub `s-maxage` przesłania ustawienia ważności strony dokonane za pomocą parametru `Expires`.

Poniższa funkcja powoduje wygenerowanie nagłówka strony, której aktualność musi być zawsze sprawdzana przez wszystkie rodzaje buforów:

```
function validate_cache_headers($my_modtime)
{
    $pretty_modtime = gmdate('D, d M Y H:i:s', $my_modtime) . ' GMT';
    if($_SERVER['IF_MODIFIED_SINCE'] == $gmt_mtime) {
        header("HTTP/1.1 304 Not Modified");
        exit;
    }
    else {
        header("Cache-Control: must-revalidate");
        header("Last-Modified: $pretty_modtime");
    }
}
```

Funkcja pobiera jako argument czas ostatniej modyfikacji strony, a następnie porównuje go z czasem określonym w parametrze `If-Modified-Since` przesłanym przez przeglądarkę klienta. Jeśli czasy są identyczne, kopia znajdująca się w buforze jest aktualna, a zatem do klienta zwracany jest kod `304`, który oznacza, że można wykorzystać kopię z bufora. W innym przypadku ustawiany jest parametr `Last-Modified` wraz z dyrektywą `Cache-Control`, nakazującą odświeżenie strony w buforze.

Aby wykorzystać tę funkcję, trzeba znać czas ostatniej modyfikacji strony. W przypadku stron statycznych (na przykład ilustracji lub prostych, niedynamicznych stron HTML) jest nim po prostu czas ostatniej modyfikacji pliku. W przypadku stron generowanych dynamicznie (z wykorzystaniem PHP lub podobnej techniki), czas ostatniej modyfikacji to moment, w którym zmieniono którykolwiek z elementów wykorzystywanych do wygenerowania strony.

Rozważmy aplikację rejestrującą zdarzenia serwera WWW, która wyświetla aktualne dane na swojej głównej stronie:

```
$dbh = new DB_MySQL_Prod ();
$result = $dbh->execute("SELECT max(timestamp)
              FROM weblog_entries") ;
if($result) {
    list($ts) = $result->fetch_row();
    validate_cache_headers($ts);
}
```

Dla tej strony czasem ostatniej modyfikacji jest znacznik czasu ostatniego zapisu.

Jeśli wiemy, że strona będzie aktualna przez jakiś czas i nie jest dla nas ważne, że czasami może być przestarzała, możemy wyłączyć ustawienie `must-revalidate` i jawnie ustawić wartość parametru `Expires`. Należy pamiętać, że w takim przypadku dane będą w pewnym sensie nieaktualne: poinformowanie bufora proxy, że serwowana treść będzie aktualna przez jakiś czas, uniemożliwia aktualizację tej treści dla wybranego klienta w wybranym przedziale czasu. W wielu przypadkach nie stanowi to żadnego problemu.

Rozważmy na przykład witrynę WWW z serwisem informacyjnym, taką jak witryna CNN. Nawet w przypadku takich stron, które zawierają najświeższe wiadomości, brak aktualizacji strony z przez jedną minutę nie stanowi wielkiego problemu. Aby uzyskać taki efekt, można ustawić parametry nagłówków na kilka sposobów.

Aby strona była buforowana przez współdzielone serwery proxy przez okres jednej minuty, można wywołać następującą funkcję:

```
function cache_novalidate($interval = 60)
{
    $now = time();
    $pretty_lmtime = gmdate('D, d M Y H:i:s', $now) . ' GMT';
    $pretty_extime = gmdate('D, d M Y H:i:s', $now + $interval) . ' GMT';
    // Zgodność wstecz z klientami HTTP/1.0
    header("Last Modified: $pretty_lmtime");
    header("Expires: $pretty_extime");
    //Obsługa HTTP/1.1
    header("Cache-Control: public,max-age=$interval");
}
```

Z kolei dla stron spersonalizowanych można ustawić możliwość buforowania wyłącznie w przeglądarce:

```
function cache_browser($interval = 60)
{
    $now = time();
    $pretty_lmtime = gmdate('D, d M Y H:i:s', $now) . ' GMT';
    $pretty_extime = gmdate('D, d M Y H:i:s', $now + $interval) . ' GMT';
    // Zgodność wstecz z klientami HTTP/1.0
    header("Last Modified: $pretty_lmtime");
    header("Expires: $pretty_extime");
    // Obsługa HTTP/1.1
    header("Cache-Control: private,max-age=$interval',s-maxage=0");
}
```

Wreszcie, w celu zablokowania buforowania strony można wprowadzić taką oto funkcję:

```
function cache_none($interval = 60)
{
  // Zgodność wstecz z klientami HTTP/1.0
  header("Expires: 0");
  header("Pragma: no-cache");
  // Obsługa HTTP/1.1
  header ("Cache-Control: no-cache, no-store, max-age=0, s-maxage=0, must-revalidate");
}
```

Mechanizm obsługi sesji w PHP ustawia pokazane powyżej nagłówki blokujące buforowanie w chwili wywołania funkcji `session_start()`. Jeśli wydaje się nam, że lepiej znamy aplikację obsługującą sesję od autorów rozszerzenia, możemy odtworzyć nagłówki po wywołaniu funkcji `session_start()`.

Poniżej podano kilka uwag, o których należy pamiętać podczas wykorzystywania mechanizmów zewnętrznego buforowania:

- Za pomocą mechanizmów tego typu nie można buforować stron żądanych przy użyciu metody `POST`.
- Zastosowanie buforowania nie oznacza, że strona jest serwowana tylko raz, a jedynie, że jest serwowana raz do określonego serwera proxy w okresie aktualności bufora.
- Nie wszystkie serwery proxy są zgodne z dokumentem RFC. W przypadku wątpliwości lepiej zachować ostrożność i nie stosować buforowania.

Kompresja

W specyfikacji HTTP 1.0 wprowadzono pojęcie kodowania zawartości, umożliwiające wysyłanie przez klienta informacji do serwera o możliwości obsługi przekazywanych stron w postaci zaszyfrowanej. W wyniku kompresji przesyłane dokumenty mają mniejsze rozmiary. Dzięki temu uzyskuje się dwa efekty:

- Zmniejszenie zapotrzebowania na pasmo ze względu na mniejszą objętość przesyłanych danych. W wielu firmach pasmo stanowi największy składnik kosztów technologii.
- Zmniejszenie opóźnień w sieci ze względu na to, że mniejsze dokumenty można podzielić na mniejszą liczbą pakietów sieciowych.

Korzyści uzyskuje się kosztem czasu procesora potrzebnego do wykonania kompresji. W wyniku przeprowadzonego przeze mnie testu skuteczności kompresji (za pomocą narzędzia *mod_gzip*) uzyskałem nie tylko mniejsze o 30 % zużycia pasma, ale także ogólny zysk wydajności — około 10% większy współczynnik liczby stron na sekundę w porównaniu z konfiguracją bez kompresji. Nawet gdyby nie udało się uzyskać ogólnego wzrostu wydajności, uzyskany efekt oszczędności, w związku z mniejszym o 30% zużyciem pasma, był imponujący.

Kiedy przeglądarka kliencka wysyła żądanie, przesyła nagłówek określający jej typ oraz obsługiwane przez nią funkcje. W nagłówku przeglądarka informuje o akceptowanych metodach kompresji w następujący sposób:

```
Content-Encoding: gzip,deflate
```

Istnieje wiele sposobów kompresji. W przypadku kompilacji PHP z obsługą biblioteki *zlib* (z opcją `-enable-zlib` w fazie kompilacji) najprostszym sposobem kompresji jest wykorzystanie wbudowanego mechanizmu obsługi kompresji *gzip*. Właściwość tę można uaktywnić poprzez ustawienie parametru w pliku *php.ini* w następujący sposób:

```
zlib.output_compression On
```

Ustawienie tej opcji powoduje automatyczne ustalenie możliwości przeglądarki wysyłającej żądanie natychmiast po sprawdzeniu nagłówka i wysyłanie dokumentów odpowiednio skompresowanych.

Wadą kompresji w PHP jest możliwość jej stosowania tylko dla stron generowanych przez PHP. Jeżeli serwer obsługuje wyłącznie strony PHP, nie ma problemu. W innym przypadku do kompresji można wykorzystać zewnętrzny moduł serwera Apache (np. *mod_deflate* lub *mod_gzip*).

Dalsze lektury

W niniejszym rozdziale zaprezentowano kilka nowych technik. Niektóre z nich są zbyt obszerne, aby można było je opisać dostatecznie szczegółowo. Poniżej wskazano opracowania, w których można znaleźć dodatkowe informacje.

Dokumenty RFC

Informacji zawsze najlepiej szukać u ich źródła. Protokoły stosowane w internecie są zdefiniowane w dokumentach *RFC* (skrót od ang. *Request for Comment* — zapytanie o opinię) wydawanych przez zespół roboczy *IETF* (skrót od ang. *Internet Engineering Task Force*). W dokumencie RFC 2616 opisano dodatkowe parametry nagłówka protokołu HTTP 1.1. Jest to miarodajne źródło składni i semantyki dla różnorodnych dyrektyw nagłówka. Dokumenty RFC można pobrać z wielu miejsc w internecie. Moim ulubionym miejscem jest witryna zespołu IETF: *www.ietf.org/rfc.html*.

Bufory kompilatora

Więcej informacji na temat działania buforów kompilatora można znaleźć w rozdziałach 21. i 24.

Nick Lindridge, autor akceleratora *ionCube*, napisał doskonały artykuł dotyczący działania stworzonego przez siebie narzędzia. Dokument jest dostępny pod adresem *www.php--accelerator.co.uk/PHPA_Article.pdf*.

Kod źródłowy bufora *APC* można znaleźć w witrynie repozytorium PEAR bibliotek PHP.

Po kod binarny akceleratora *ionCube* odsyłam pod adres *www.ioncube.com*.

Akcelerator *Zend* znajduje się pod adresem *www.zend.com*.

Bufory proxy

Serwer Squid jest dostępny pod adresem *www.squid-cache.org*. W witrynie znajduje się także wiele doskonałych zasobów dotyczących konfiguracji i zastosowań serwera. Interesujący artykuł dotyczący zastosowania serwera Squid jako akceleratora HTTP można przeczytać w witrynie *ViSolve* pod adresem *http://squid.visolve.com/white_papers/reverseproxy.htm*. Dodatkowe materiały poświęcone poprawie wydajności serwera Squid jako odwrotnego serwera proxy można znaleźć pod adresem *http://squid.sourceforge.net/rproxy*.

Po moduł *mod_backhand* odsyłam pod adres *http://www.backhand.org/mod_backhand/*.

W niniejszym rozdziale przedstawiono zaledwie podstawowy opis zastosowania modułu *mod_proxy*. Doskonałe efekty obsługi żądań można uzyskać poprzez integrację modułu *mod_proxy* z modułem *mod_rewrite*.

Więcej informacji można znaleźć na witrynie WWW projektu Apache (*http://www.apache.org*). Zwięzły przykład integracji modułów *mod_rewrite* i *mod_proxy* przedstawiłem w mojej prezentacji na konferencji Apachecon w 2002 r. (*Scalable Internet Architectures* — skalowalne architektury internetowe). Slajdy z tej prezentacji są dostępne pod adresem *http://www.omniti.com/~george/talks/LV736.ppt*.

Moduł *mod_accel* można znaleźć pod adresem *http://sysoev.ru/mod_accel*. Niestety, większa część dokumentacji jest po rosyjsku. Napisany w języku angielskim przewodnik *how-to* autorstwa Phillipa Maka dotyczący instalacji modułów *mod_accel* i *mod_deflate* znajduje się pod adresem *http://www.aaanime.net/pmak/apache/mod_accel*.

Kompresja

Po moduł *mod_deflate* dla serwera Apache w wersji *1.3.x* odsyłam pod adres *http://sysoev.ru/mod_deflate*. Moduł ten nie ma nic wspólnego z modułem *mod_deflate* dla serwera Apache 2.0. Podobnie jak w przypadku modułu *mod_accel*, dokumentacja tego projektu jest niemal w całości po rosyjsku.

Moduł *mod_gzip* opracowała firma Remote Communications, ale obecnie ma on nową lokalizację na witrynie *Sourceforge*, pod adresem *http://sourceforge.net/projects/mod-gzip*.