helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

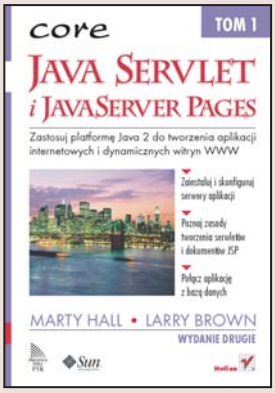
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Java Servlet i JavaServer Pages. Tom 1. Wydanie II



Autorzy: Marty Hall, Larry Brown
Tłumaczenie: Piotr Rajca
ISBN: 83-246-0032-9
Tytuł oryginału: Core Servlets and JavaServer Pages, Vol. 1: Core Technologies, Second Edition
Format: B5, stron: 640

Zastosuj platformę Java 2 do tworzenia aplikacji internetowych i dynamicznych witryn WWW

- Zainstaluj i skonfiguruj serwery aplikacji
- Poznaj zasady tworzenia serwletów i dokumentów JSP
- Połącz aplikację z bazą danych

Java Servlet i JavaServer Pages to oparte na platformie Java 2 technologie wykorzystywane do tworzenia aplikacji internetowych i dynamicznych witryn WWW. Stosuje się je wszędzie tam, gdzie niezbędna jest wysoka stabilność i niezawodność, wydajność przy przetwarzaniu danych i elastyczność pozwalająca na szybkie dodawanie kolejnych modułów rozszerzających możliwości aplikacji. W oparciu o te technologie powstają między innymi systemy bankowości elektronicznej, aplikacje finansowe oraz systemy do obsługi programów lojalnościowych. Ogromną zaletą tej technologii jest fakt, że zarówno narzędzia programistyczne, jak i serwery aplikacji dostępne są nieodpłatnie, na zasadach open source, co redukuje koszty tworzenia oraz użytkowania stworzonych za ich pomocą systemów.

Książka „Java Servlet i JavaServer Pages. Tom 1. Wydanie II" przedstawia sposoby wykorzystywania najnowszych możliwości serwletów i stron JSP. Wyjaśnia, w jakich przypadkach należy stosować serwlety, w jakich JSP oraz kiedy połączyć te technologie. Czytając ją, poznasz specyfikację technologii Java Servlet w wersji 2.4 oraz technologii JSP w wersji 2.0, nauczysz się instalować i konfigurować serwery aplikacji oraz dowiesz się, jak tworzyć aplikacje w obu technologiach i jak łączyć je z bazami danych za pomocą interfejsów JDBC. Zastosujesz w swoich projektach najlepsze strategie, których omówienie również znajdziesz w tej książce.

- Instalacja i konfiguracja serwera Tomcat, JRun i Resin
- Zasady tworzenia serwletów
- Obsługa protokołu HTTP oraz danych z formularzy HTML
- Generowanie plików XLS
- śledzenie sesji i stosowanie plików cookie
- Podstawy stosowania JSP
- Wykorzystywanie komponentów JavaBean
- Architektura MVC
- Połączenie z bazami danych za pomocą JDBC

Napisz wydajne i stabilne aplikacje internetowe, wykorzystując nowoczesne technologie

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

Spis treści

Podziękowania 13

O autorach 14

Wprowadzenie 15

Rozdział 1. Ogólne informacje o serwletach i JavaServer Pages 25

1.1. Rola serwletów 25
1.2. Dlaczego warto dynamicznie generować strony WWW? 27
1.3. Przedstawienie kodu serwletu 28
1.4. Zalety serwletów w porównaniu ze zwykłymi programami CGI 30
Efektywność 30
Wygoda 30
Duże możliwości 31
Przenośność 31
Niewielkie koszty 31
Bezpieczeństwo 32
Główny kierunek rozwoju 33
1.5. JavaServer Pages 33

Część I Serwlety 35

Rozdział 2. Instalacja i konfiguracja serwera 37

2.1. Pobranie oraz instalacja Java Software Development Kit (SDK) 38
2.2. Pobranie serwera na lokalny komputer 39
2.3. Konfiguracja serwera 42
2.4. Konfiguracja serwera Tomcat 43
Określanie wartości zmiennej środowiskowej JAVA_HOME 44
Określanie portu używanego przez serwer 45
Uaktywnianie opcji automatycznej aktualizacji serwletów 46
Uaktywnienie kontekstu głównego 47
Włączenie serwletu wywołującego 47
Powiększenie limitów pamięci dla programów DOS 48
Określenie wartości zmiennej środowiskowej CATALINA_HOME 48
Testowanie podstawowej konfiguracji serwera 48

2.5. Konfiguracja serwera JRun ... 49
Numer seryjny serwera ... 50
Ograniczenia użytkowników ... 50
Katalog instalacyjny Javy ... 50
Katalog instalacyjny serwera ... 51
Nazwa i hasło administratora ... 51
Możliwość automatycznego uruchamiania serwera ... 51
Określenie portu serwera ... 52
Testowanie podstawowej konfiguracji serwera ... 53
2.6. Konfiguracja serwera Resin ... 54
Określanie wartości zmiennej środowiskowej JAVA_HOME ... 54
Określenie portu używanego przez serwer Resin ... 55
Testowanie podstawowej konfiguracji serwera ... 55
2.7. Konfiguracja środowiska pracy ... 55
Stworzenie katalogu roboczego ... 56
Określanie wartości zmiennej środowiskowej CLASSPATH ... 57
Tworzenie skrótów ułatwiających uruchamianie i zatrzymywanie serwera ... 58
Zapisanie adresu lub zainstalowanie dokumentacji API serwletów i JSP ... 59
2.8. Testowanie konfiguracji ... 60
Sprawdzenie konfiguracji SDK ... 60
Sprawdzenie podstawowej konfiguracji serwera ... 61
Kompilacja i wdrożenie prostych serwletów ... 63
2.9. Tworzenie uproszczonej metody wdrażania ... 69
Utworzenie skrótów lub łączy symbolicznych ... 70
Stosowanie opcji -d kompilatora ... 71
Wdrażanie przy wykorzystaniu IDE ... 71
Zastosowanie programu ant lub podobnego ... 72
2.10. Katalogi wdrożeniowe dla domyślnych aplikacji WWW. Podsumowanie ... 72
Tomcat ... 73
JRun ... 73
Resin ... 74
2.11. Aplikacje WWW — prezentacja ... 75
Utworzenie katalogu aplikacji ... 76
Aktualizacja zmiennej środowiskowej CLASSPATH ... 77
Zarejestrowanie aplikacji WWW na serwerze ... 77
Stosowanie prefiksów adresów URL ... 79
Przypisywanie serwletom niestandardowych adresów URL ... 81

Rozdział 3. Podstawy tworzenia serwletów ... 85

3.1. Podstawowa struktura serwletów ... 86
3.2. Serwlet generujący zwykły tekst ... 88
3.3. Serwlet generujący kod HTML ... 89
3.4. Umieszczanie serwletów w pakietach ... 92
3.5. Proste narzędzia pomocne podczas tworzenia dokumentów HTML ... 93
3.6. Cykl istnienia serwletów ... 96
Metoda service ... 97
Metody doGet, doPost oraz doXxx ... 98
Metoda init ... 98
Metoda destroy ... 103
3.7. Interfejs SingleThreadModel ... 104
3.8. Testowanie serwletów ... 108

Rozdział 4. Obsługa żądań: dane przesyłane za pomocą formularzy 111

4.1. Znaczenie informacji przesyłanych za pomocą formularzy 111
4.2. Odczytywanie danych formularzy w serwletach 113
Odczytywanie pojedynczych wartości: metoda getParameter 113
Odczytywanie wielu wartości: metoda getParameterValues 114
Pobieranie nazw parametrów: metody getParameterNames oraz getParameterMap 114
Odczytywanie nieprzetworzonych danych formularza i analiza przesyłanych plików: metody getReader oraz getInputStream 115
Odczyt danych zakodowanych przy użyciu różnych zbiorów znaków: metoda setCharacterEncoding 116
4.3. Przykład: odczytanie trzech konkretnych parametrów 117
4.4. Przykład: odczytanie wszystkich parametrów 120
4.5. Stosowanie wartości domyślnych, gdy nie podano parametru lub jego wartość jest niewłaściwa 123
4.6. Filtrowanie łańcuchów w poszukiwaniu znaków specjalnych HTML 133
Implementacja filtrowania 134
Przykład: serwlet wyświetlający fragmenty kodu 136
4.7. Automatyczne zapisywanie wartości parametrów w obiektach Javy: komponenty formularzy 139
Zastosowanie klasy BeanUtilities 142
Pobieranie i instalacja pakietów Jakarta Commons 145
4.8. Ponowne wyświetlanie formularza w przypadku pominięcia lub błędnego podania wartości parametrów 146
Opcje ponownego wyświetlania formularza 146
Serwlet obsługujący aukcje 148

Rozdział 5. Obsługa żądań: nagłówki żądań HTTP 155

5.1. Odczytywanie wartości nagłówków żądania w serwletach 156
5.2. Wyświetlanie wszystkich nagłówków 157
5.3. Nagłówki żądań protokołu HTTP 1.1 159
5.4. Przesyłanie skompresowanych stron WWW 164
5.5. Rozróżnianie przeglądarek różnych typów 167
5.6. Modyfikowanie strony w zależności od tego, skąd użytkownik na nią trafił 169
5.7. Dostęp do standardowych zmiennych CGI 173
Odpowiedniki zmiennych CGI dostępne w serwletach 173
Serwlet wyświetlający wartości zmiennych CGI 176

Rozdział 6. Generowanie odpowiedzi: kody statusu HTTP 179

6.1. Określanie kodów statusu 180
Określanie dowolnego kodu statusu: metoda setStatus 180
Stosowanie kodów statusu 302 oraz 404: metody sendRedirect oraz sendError 181
6.2. Kody statusu protokołu HTTP 1.1 oraz ich przeznaczenie 182
6.3. Serwlet przekierowujący użytkownika na stronę wybraną zależnie od używanej przeglądarki 187
6.4. Interfejs użytkownika obsługujący różne serwisy wyszukujące 189

Rozdział 7. Generowanie odpowiedzi: nagłówki odpowiedzi HTTP 195

7.1. Określanie nagłówków odpowiedzi z poziomu serwletów 195
7.2. Nagłówki odpowiedzi protokołu HTTP 1.1 oraz ich znaczenie 197
7.3. Generowanie arkuszy kalkulacyjnych programu Excel 204

7.4. Trwałe przechowywanie stanu serwletu i automatyczne odświeżanie stron 205
Znajdowanie liczb pierwszych na potrzeby systemów kryptograficznych z kluczem publicznym 206
7.5. Zastosowanie serwletów do generowania obrazów JPEG 215

Rozdział 8. Obsługa cookies 225

8.1. Korzyści płynące ze stosowania cookies 225
Identyfikacja użytkowników podczas trwania sesji na witrynach zajmujących się handlem elektronicznym 226
Zapamiętywanie nazwy użytkownika i hasła 226
Dostosowywanie witryny 227
Dobór reklam 227
8.2. Niektóre problemy związane ze stosowaniem cookies 227
8.3. Usuwanie cookies 230
8.4. Wysyłanie i odbieranie cookies 230
Wysyłanie cookies do przeglądarki 231
Odczytywanie cookies przesyłanych przez klienta 233
8.5. Zastosowanie cookies do wykrywania użytkowników odwiedzających stronę po raz pierwszy 234
8.6. Stosowanie atrybutów cookies 236
8.7. Rozróżnianie cookies sesyjnych od trwałych 239
8.8. Proste narzędzia do obsługi cookies 242
Odnajdywanie cookie o określonej nazwie 242
Tworzenie cookies o długim czasie ważności 243
8.9. Praktyczne zastosowanie klas ułatwiających obsługę cookies 244
8.10. Modyfikacja wartości cookie: śledzenie ilości odwiedzin użytkownika 246
8.11. Użycie cookies do zapamiętywania preferencji użytkownika 248

Rozdział 9. Śledzenie sesji 253

9.1. Potrzeba śledzenia sesji 253
Cookies 254
Przepisywanie adresów URL 254
Ukryte pola formularzy 255
Śledzenie sesji w serwletach 255
9.2. Podstawowe informacje o śledzeniu sesji 255
Pobieranie obiektu HttpSession skojarzonego z bieżącym żądaniem 256
Pobieranie informacji skojarzonych z sesją 257
Kojarzenie informacji z sesją 257
Usuwanie danych sesji 258
9.3. Interfejs programowania aplikacji służący do obsługi sesji 258
9.4. Sesje przeglądarki a sesje serwera 260
9.5. Przepisywanie adresów URL przesyłanych do przeglądarki 261
9.6. Serwlet generujący indywidualny licznik odwiedzin dla każdego użytkownika 263
9.7. Gromadzenie listy danych użytkownika 265
9.8. Internetowy sklep wykorzystujący koszyki i śledzenie sesji 269
Tworzenie interfejsu użytkownika 269
Obsługa zamówień 273
To, czego nie widać: implementacja koszyka i katalogu produktów 277

Część II JavaServer Pages 285

Rozdział 10. Przegląd technologii JSP ... 287

10.1. Potrzeba technologii JSP ... 288
10.2. Zalety JSP ... 289
10.3. Zalety JSP w porównaniu z innymi technologiami ... 290
W porównaniu z platformą .NET oraz technologią Active Server Pages (ASP) ... 291
W porównaniu z PHP ... 291
W porównaniu z serwletami ... 291
W porównaniu z językiem JavaScript ... 292
W porównaniu z WebMacro oraz Velocity ... 292
10.4. Błędne opinie dotyczące JSP ... 293
Ignorowanie faktu, że JSP jest technologią działającą po stronie serwera ... 293
Mylenie czasu przekształcania z czasem żądania ... 294
Przypuszczenie, że wystarczy znajomość i stosowanie samej technologii JSP ... 295
Przypuszczenie, że wystarczy znajomość i stosowanie samych serwletów ... 296
10.5. Instalacja stron JSP ... 296
Katalogi JSP na serwerze Tomcat (domyślna aplikacja WWW) ... 297
Katalogi JSP na serwerze JRun (domyślna aplikacja WWW) ... 297
Katalogi JSP na serwerze Resin (domyślna aplikacja WWW) ... 297
10.6. Podstawowa składnia JSP ... 298
Tekst HTML ... 298
Komentarze HTML ... 298
Tekst szablonu ... 298
Komentarze JSP ... 299
Wyrażenia JSP ... 299
Skryptlet ... 299
Deklaracja JSP ... 299
Dyrektywa JSP ... 300
Akcja JSP ... 300
Element Języka Wyrażeń JSP ... 300
Znacznik niestandardowy (akcja niestandardowa) ... 300
Zabezpieczony tekst szablonu ... 301

Rozdział 11. Wywoływanie kodu Javy przy użyciu elementów skryptowych JSP ... 303

11.1. Tworzenie tekstu szablonu ... 303
11.2. Wywoływanie kodu Javy w stronach JSP ... 304
Typy elementów skryptowych JSP ... 305
11.3. Ograniczanie ilości kodu umieszczanego na stronach JSP ... 305
Znaczenie stosowania pakietów ... 307
11.4. Stosowanie wyrażeń JSP ... 309
Zmienne predefiniowane ... 309
Te same rozwiązania w serwletach i stronach JSP ... 309
Zapis wyrażeń przy użyciu składni XML ... 311
11.5. Przykład: wyrażenia JSP ... 311
11.6. Porównanie serwletów i stron JSP ... 313
11.7. Pisanie skryptletów ... 315
Te same rozwiązania w serwletach i stronach JSP ... 316
Składnia XML do zapisu skryptletów ... 317
11.8. Przykład skryptletu ... 317
11.9. Zastosowanie skryptletów do warunkowego wykonywania fragmentów stron JSP ... 319

11.10. Stosowanie deklaracji ... 321
Te same rozwiązania w serwletach i stronach JSP ... 322
Składnia XML do zapisu deklaracji ... 323
11.11. Przykład zastosowania deklaracji ... 323
11.12. Stosowanie zmiennych predefiniowanych ... 325
11.13. Porównanie wyrażeń JSP, deklaracji oraz skryptletów ... 327
Przykład 1.: wyrażenia JSP ... 328
Przykład 2.: skryptlety ... 328
Przykład 3.: deklaracje JSP ... 330

Rozdział 12. Dyrektywa page: strukturalizacja generowanych serwletów ... 333

12.1. Atrybut import ... 334
12.2. Atrybuty contentType oraz pageEncoding ... 337
Generowanie arkuszy kalkulacyjnych programu Microsoft Excel ... 337
12.3. Warunkowe generowanie arkuszy kalkulacyjnych programu Microsoft Excel ... 338
12.4. Atrybut session ... 341
12.5. Atrybut isELIgnored ... 341
12.6. Atrybuty buffer oraz autoFlush ... 342
12.7. Atrybut info ... 343
12.8. Atrybuty errorPage oraz isErrorPage ... 343
12.9. Atrybut isThreadSafe ... 346
12.10. Atrybut extends ... 347
12.11. Atrybut language ... 348
12.12. Składnia XML zapisu dyrektyw ... 348

Rozdział 13. Dołączanie plików i apletów do dokumentów JSP ... 349

13.1. Dołączanie plików podczas obsługi żądań: znacznik akcji jsp:include ... 350
Atrybut page: określanie dołączanej strony ... 350
Składnia XML a znacznik akcji jsp:include ... 352
Atrybut flush ... 352
Strona z najnowszymi doniesieniami ... 352
Element jsp:param: uzupełnianie parametrów żądania ... 354
13.2. Dołączanie plików w czasie przekształcania strony ... 355
Problemy z utrzymaniem kodu w przypadku stosowania dyrektywy include ... 356
Dodatkowe możliwości, jakie daje dyrektywa include ... 357
Aktualizacja strony głównej ... 357
Zapis dyrektywy include w postaci elementu XML ... 358
Przykład: wielokrotne stosowanie stopki ... 358
13.3. Przekazywanie żądań przy użyciu znacznika akcji jsp:forward ... 360
13.4. Dołączanie apletów związanych z Java Plug-In ... 361
Element jsp:plugin ... 363
Elementy jsp:param oraz jsp:params ... 365
Element jsp:fallback ... 365
Przykład zastosowania znacznika jsp:plugin ... 366

Rozdział 14. Zastosowanie komponentów JavaBean w dokumentach JSP ... 371

14.1. Dlaczego warto stosować komponenty? ... 371
14.2. Czym są komponenty? ... 372
14.3. Podstawowe sposoby użycia komponentów ... 374
Tworzenie komponentów: jsp:useBean ... 374
Instalacja klas komponentów ... 375

Stosowanie atrybutów znacznika akcji jsp:useBean: scope, beanName oraz type ... 375
Dostęp do właściwości komponentów: jsp:getProperty ... 376
Podstawowy sposób określania wartości właściwości: jsp:setProperty ... 377
14.5. Określanie wartości właściwości komponentów: techniki zaawansowane ... 380
Kojarzenie właściwości z parametrami wejściowymi ... 384
Kojarzenie wszystkich właściwości z parametrami wejściowymi ... 385
14.6. Współużytkowanie komponentów ... 386
Warunkowe tworzenie komponentów ... 387
14.7. Współużytkowanie komponentów na cztery różne sposoby — przykład ... 390
Tworzenie komponentu oraz testera komponentów ... 391
Zastosowanie atrybutu scope="page" — komponent nie jest współużytkowany 392
Współużytkowanie komponentu w obrębie żądania ... 393
Współużytkowanie komponentu w obrębie sesji ... 396
Współużytkowanie komponentu w obrębie aplikacji ... 398

Rozdział 15. Integracja serwletów i JSP: architektura model-widok-kontroler (MVC) ... 401

15.1. Potrzeba stosowania architektury MVC ... 401
Szkielety MVC ... 403
Architektura czy podejście? ... 403
15.2. Implementacja architektury MVC przy użyciu interfejsu RequestDispatcher ... 403
Definiowanie komponentów reprezentujących dane ... 404
Tworzenie serwletów obsługujących żądania ... 404
Zapis danych w komponentach ... 405
Zapis wyników ... 405
Przekazywanie żądania na stronę JSP ... 406
Pobieranie danych z komponentów ... 408
15.3. Podsumowanie kodu architektury MVC ... 409
Przechowywanie współużytkowanych danych w obiekcie żądania ... 409
Przechowywanie współużytkowanych danych w obiekcie sesji ... 409
Przechowywanie współużytkowanych danych w obiekcie aplikacji ... 410
15.4. Interpretacja adresów URL na stronie docelowej ... 410
15.5. Stosowanie architektury MVC: dostęp do stanu konta bankowego ... 411
15.6. Porównanie trzech sposobów współużytkowania danych ... 417
Współużytkowanie danych przy użyciu obiektu żądania ... 418
Współużytkowanie danych przy użyciu obiektu sesji ... 419
Współużytkowanie danych przy użyciu kontekstu serwletu ... 422
15.7. Przekazywanie żądań ze stron JSP ... 424
15.8. Dołączanie stron ... 425

Rozdział 16. Upraszczanie dostępu do kodu Javy: język wyrażeń JSP 2.0 ... 427

16.1. Cele stosowania języka wyrażeń ... 427
16.2. Stosowanie języka wyrażeń ... 429
Oznaczanie znaków specjalnych ... 430
16.3. Zapobieganie przetwarzaniu wyrażeń EL ... 430
Wyłączanie obsługi języka wyrażeń w całej aplikacji ... 430
Wyłączanie obsługi języka wyrażeń na wielu stronach JSP ... 431
Wyłączanie obsługi języka wyrażeń na konkretnych stronach JSP ... 431
Wyłączanie obsługi konkretnych instrukcji języka wyrażeń ... 432
16.4. Uniemożliwianie stosowania zwyczajnych elementów skryptowych ... 432

16.5. Odwołania do zmiennych dostępnych w określonym zakresie ... 433
Wybór nazw atrybutów ... 434
Przykład ... 434
16.6. Dostęp do właściwości komponentów ... 436
Równoważność notacji kropkowej oraz notacji charakterystycznej dla tablic ... 437
Przykład ... 438
16.7. Dostęp do zawartości kolekcji ... 441
Przykład ... 441
16.8. Odwołania do obiektów niejawnych ... 443
Przykład ... 445
16.9. Stosowanie operatorów języka wyrażeń ... 446
Operatory arytmetyczne ... 446
Operatory relacyjne ... 447
Operatory logiczne ... 448
Operator empty ... 448
Przykład ... 448
16.10. Warunkowe przetwarzanie wyrażeń ... 450
Przykład ... 451
16.11. Informacje o innych możliwościach języka wyrażeń ... 453

Część III Technologie pomocnicze 455

Rozdział 17. Obsługa baz danych przy użyciu JDBC ... 457

17.1. Podstawowe sposoby stosowania JDBC ... 458
Załadowanie sterownika ... 459
Określenie adresu URL połączenia ... 461
Nawiązanie połączenia ... 461
Utworzenie polecenia ... 462
Wykonanie zapytania ... 462
Przetworzenie wyników ... 463
Zamknięcie połączenia ... 466
17.2. Prosty przykład wykorzystania JDBC ... 466
17.3. Narzędzia JDBC ułatwiające korzystanie z baz danych ... 472
17.4. Stosowanie poleceń przygotowanych ... 483
17.5. Tworzenie poleceń wywoływanych ... 487
Zdefiniowanie wywołania procedury zachowanej ... 488
Przygotowanie obiektu CallableStatement reprezentującego procedurę ... 488
Zarejestrowanie typów parametrów wyjściowych ... 489
Podanie wartości parametrów wejściowych ... 489
Wykonanie procedury zachowanej ... 489
Pobranie parametrów wyjściowych ... 489
Przykład ... 490
17.6. Stosowanie transakcji ... 493
17.7. Odwzorowywanie danych na obiekty przy użyciu szkieletów ORM ... 497

Rozdział 18. Konfiguracja baz danych MS Access, MySQL oraz Oracle9i ... 505

18.1. Konfiguracja programu Microsoft Access do współpracy z JDBC ... 506
Wybór systemowej nazwy źródła danych
w programie Administrator źródeł danych ODBC ... 507
Wybór sterownika dla DSN ... 507
Wybór źródła danych ... 508
Zaakceptowanie nowego DSN ... 509

18.2. Instalacja i konfiguracja serwera MySQL ... 509
Pobieranie i instalacja serwera MySQL ... 510
Tworzenie bazy danych ... 510
Tworzenie użytkownika ... 511
Instalacja sterownika JDBC ... 511
18.3. Instalacja i konfiguracja serwera Oracle9i ... 512
Pobieranie i instalacja Oracle9i ... 514
Tworzenie bazy danych ... 522
Tworzenie bazy danych przy użyciu programu Configuration Assistant ... 522
Ręczne tworzenie bazy danych ... 528
Utworzenie użytkownika ... 535
Instalacja sterownika JDBC ... 535
18.4. Testowanie bazy danych przy wykorzystaniu połączenia JDBC ... 536
18.5. Tworzenie tabeli music ... 543
Tworzenie tabeli music przy użyciu programu CreateMusicTable.java ... 543
Tworzenie tabeli music przy użyciu skryptu create_music_table.sql ... 546

Rozdział 19. Tworzenie i przetwarzanie formularzy HTML ... 549

Domyślna aplikacja WWW: Tomcat ... 549
Domyślna aplikacja WWW: JRun ... 550
Domyślna aplikacja WWW: Resin ... 550
19.1. Jak przesyłane są dane z formularzy HTML ... 550
19.2. Element FORM ... 554
19.3. Tekstowe elementy sterujące ... 560
Pola tekstowe ... 560
Pola hasła ... 562
Wielowierszowe pola tekstowe ... 563
19.4. Przyciski ... 564
Przycisk SUBMIT ... 565
Przyciski RESET ... 567
Przyciski JavaScript ... 568
19.5. Pola wyboru i przyciski opcji ... 569
Pola wyboru ... 569
Przyciski opcji ... 570
19.6. Listy i listy rozwijane ... 572
19.7. Element sterujący służący do przesyłania plików ... 576
19.8. Mapy odnośników obsługiwane na serwerze ... 578
IMAGE — standardowe mapy odnośników obsługiwane po stronie serwera ... 579
ISMAP — alternatywny sposób tworzenia map odnośników obsługiwanych po stronie serwera ... 580
19.9. Pola ukryte ... 582
19.10. Grupowanie elementów sterujących ... 583
19.11. Określanie kolejności poruszania się pomiędzy elementami formularzy ... 585
19.12. Testowy serwer WWW ... 586
EchoServer ... 586

Dodatki 593

Dodatek A Organizacja i struktura serwera ... 595

Skorowidz ... 609

16

Upraszczanie dostępu do kodu Javy: język wyrażeń JSP 2.0

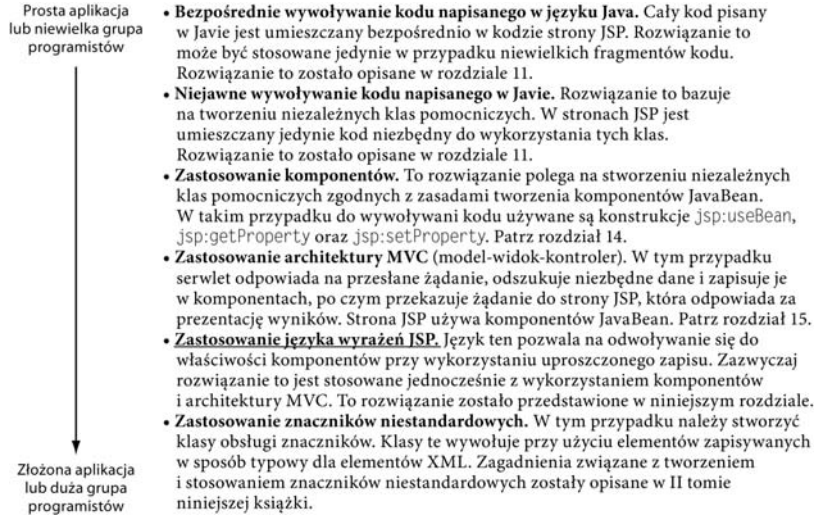
Specyfikacja JSP 2.0 wprowadza uproszczony język służący do przetwarzania i wyświetlania wartości obiektów Javy przechowywanych w standardowych „miejscach". Ten język wyrażeń (ang. *Expression Language*, nazywany także skrótowo „EL") jest jedną z najważniejszych nowych możliwości, które pojawiły się w specyfikacji JSP 2.0; kolejną jest możliwość tworzenia znaczników niestandardowych przy użyciu kodu JSP, a nie kodu Javy (zagadnienia związane z tworzeniem znaczników niestandardowych zostały opisane w II tomie niniejszej książki).

> Języka wyrażeń nie można stosować na serwerach zgodnych ze specyfikacją JSP w wersji 1.2 lub wcześniejszą.

16.1. Cele stosowania języka wyrażeń

Zgodnie z informacjami podanymi na rysunku 16.1, strona JSP może używać kodu napisanego w języku Java na kilka różnych sposobów. Jednym z najlepszych i najbardziej elastycznych rozwiązań jest architektura MVC (opisana w rozdziale 15.). W tym rozwiązaniu na żądanie przesłane przez klienta odpowiada serwlet. Wywołuje on następnie kod implementujący logikę biznesową lub realizujący operacje na bazie danych, umieszcza wynikowe dane w komponentach, które z kolei są zapisywane w obiekcie żądania, sesji lub kontekście serwletu; w końcu serwlet wywołuje stronę JSP, której zadaniem jest przedstawienie wyników.

Jedyną częścią tego rozwiązania, która utrudnia jego stosowanie, jest ostatnia wykonywana czynność: wyświetlenie wyników na stronie JSP. Zazwyczaj do tego celu są stosowane znaczniki akcji `jsp:useBean` oraz `jsp:getProperty`; jednak kod, jaki należy stworzyć, by ich użyć, jest dosyć rozbudowany i niewygodny. Co więcej, znacznik `jsp:getProperty` po zwala



Rysunek 16.1. *Strategie wywoływania dynamicznego kodu ze stron JSP*

jedynie na pobieranie wartości prostych właściwości; jeśli właściwość jest kolekcją lub innym komponentem, to odczytanie jej wartości wymaga zastosowania złożonego wyrażenia napisanego w Javie (a przecież zastosowanie architektury MVC ma na celu uniknięcie takiej konieczności).

Język wyrażeń JSP 2.0 pozwala na uproszczenie warstwy prezentacji aplikacji WWW, poprzez zastąpienie trudnego do utrzymania kodu składającego się z elementów skryptowych JSP lub znaczników akcji `jsp:useBean` oraz `jsp:getProperty` znacznie krótszymi i bardziej przejrzystymi wyrażeniami o postaci:

```
${wyrażenie}
```

Konkretnie rzecz biorąc, język wyrażeń zapewnia następujące możliwości:

- **Zwięzły sposób dostępu do zapisanych obiektów.** W celu wyświetlenia zmiennej o nazwie `sprzedazWartosc` dostępnej w określonym zakresie (czyli obiektu zapamiętanego poprzez wywołanie metody `setAttribute` obiektów `PageContext`, `HttpServletRequest`, `HttpSession` lub `ServletContext`), można użyć wyrażenia o postaci `${sprzedazWartosc}`. Patrz podrozdział 16.5
- **Uproszczony zapis odwołań do właściwości komponentów.** Aby wyświetlić właściwość `nazwaFirmy` (na przykład wynik zwracany przez metodę `getNazwaFirmy`) innej zmiennej dostępnej w określonym zakresie, należy posłużyć się wyrażeniem `${firma.nazwaFirmy}`. Podobnie, aby odczytać właściwość `imie` obiektu dostępnego jako właściwość `prezes` obiektu `firma` przechowywanego w zmiennej dostępnej w określonym zakresie, należałoby użyć wyrażenia o postaci `${firma.prezes.imie}`). Patrz podrozdział 16.6.
- **Uproszczony dostęp do elementów kolekcji.** Do odczytywania wartości elementów list, zawartości obiektów `List` oraz `Map` służą wyrażenia o postaci: `${zmienna[indeksLubKlucz]}`. Jeśli indeks lub klucz ma postać, jaką w języku Java muszą przybierać zmienne, to oprócz zapisu wykorzystującego nawiasy kwadratowe, można także zastosować standardową postać odwołań do właściwości komponentów. Patrz podrozdział 16.7.

- **Zwięzły dostęp do parametrów żądania, cookies oraz innych danych przesyłanych w żądaniu.** W celu uzyskania dostępu do standardowych rodzajów informacji przesyłanych w żądaniu, można posłużyć się kilkoma predefiniowanymi, niejawnymi obiektami. Patrz podrozdział 16.8.
- **Niewielki, lecz użyteczny zbiór prostych operatorów.** Możliwe jest wykonywanie operacji na obiektach używanych w wyrażeniach EL. Do tego celu służy kilka operatorów arytmetycznych, relacyjnych i logicznych oraz operator sprawdzający, czy została określona wartość właściwości.
- **Prezentację warunkową.** W celu wybrania jednej z opcji prezentacji, nie trzeba już stosować elementów skryptowych. Zamiast tego można użyć konstrukcji o postaci `${test ? wartosc1 : wartosc2}`. Patrz podrozdział 16.10.
- **Automatyczną konwersję typów.** Dzięki językowi wyrażeń nie trzeba stosować większości operacji rzutowania typów oraz konwersji łańcuchów znaków na liczby.
- **Zastosowanie pustych wartości zamiast komunikatów o błędach.** W większości przypadków brak wartości parametru wejściowego lub zgłoszenie wyjątku `NullPointerException` powoduje zastosowanie pustego łańcucha znaków, a nie zgłoszenie wyjątku.

16.2. Stosowanie języka wyrażeń

W JSP 2.0 język wyrażeń jest „wywoływany" w przypadku zastosowania elementu o następującej postaci:

```
${wyrażenie}
```

Elementy EL można umieszczać w zwyczajnym tekście, bądź też w atrybutach znaczników JSP (zakładając, że można w nich umieszczać standardowe wyrażenia JSP. Oto przykład:

```
<UL>
<LI>Imię: ${wyrażenie}
<LI>Adres: ${wyrażenie2}
</UL>
<jsp:include page="${expression}" />
```

W przypadku stosowania języka wyrażeń w atrybutach znaczników, można w nich umieszczać większą liczbę wyrażeń (a nawet umieszczać pomiędzy wyrażeniami fragmenty tekstu) — takie wyrażenia zostaną przetworzone do postaci łańcucha znaków, a wszystkie te łańcuchy połączone. Na przykład:

```
<jsp:include page="${wyrażenie1}NieNieNie${wyrażenie1}" />
```

W niniejszym rozdziale zostało przedstawione zastosowanie języka wyrażeń w zwyczajnym tekście. Natomiast w II tomie książki podano informacje o stosowaniu elementów EL w atrybutach samodzielnie tworzonych znaczników niestandardowych oraz znaczników wchodzących w skład bibliotek JSTL (ang. *JSP Standard Tag Library*) i JSF (ang. *JavaServer Faces*).

Oznaczanie znaków specjalnych

Jeśli się zdarzy, że w tekście strony JSP musi się pojawić łańcuch znaków `${`, to należy go zapisać jako `\${`. Jeśli w wyrażeniu EL należy umieścić apostrof lub znaku cudzysłowu, należy je zapisywać w postaci: `\'` oraz `\"`.

16.3. Zapobieganie przetwarzaniu wyrażeń EL

W specyfikacji JSP 1.2 oraz wcześniejszych, łańcuchy znaków o postaci `${...}` nie miały żadnego szczególnego znaczenia. A zatem może się zdarzyć, iż w poprawnych stronach JSP, które obecnie zaczęły być używane na serwerze obsługującym JSP 2.0, pojawią się łańcuchy znaków `${`. W takim przypadku należy wyłączyć na stronie obsługę języka wyrażeń. Można to zrobić na cztery sposoby:

- **Wyłączając obsługę języka wyrażeń w całej aplikacji WWW.** Można to zrobić w pliku *web.xml* definiującym serwlety zgodne ze specyfikacją JavaServlet w wersji 2.3 (JSP 1.2) lub wcześniejszej. Szczegółowe informacje na ten temat podano w kolejnym punkcie rozdziału.
- **Wyłączając obsługę języka wyrażeń na wielu stronach JSP.** Wybrane strony można wskazać w elemencie `jsp-property-group` umieszczanym w pliku *web.xml*. To rozwiązanie opisano w drugim kolejnym punkcie rozdziału.
- **Wyłączając obsługę języka wyrażeń na pojedynczych stronach JSP.** Do tego celu służy atrybut `isELEnabled` dyrektywy `page`. Więcej informacji na ten temat podano w trzecim kolejnym punkcie rozdziału.
- **Wyłączając pojedyncze instrukcje języka wyrażeń.** Jeśli strony JSP tworzone zgodnie ze specyfikacją 1.2 muszą być przenoszone w oryginalnej postaci (bez wprowadzania jakichkolwiek modyfikacji w pliku *web.xml*) na serwery obsługujące nowsze wersje specyfikacji JSP, to wszystkie znaki `$` występujące w normalnym tekście można zastąpić symbolem HTML o postaci `$`. W stronach JSP tworzonych zgodnie ze specyfikacją JSP 2.0, zawierających zarówno wyrażenia EL, jak i znaki `$` występujące w normalnym tekście, łańcuchy znaków `${`, które mają być traktowane dosłownie, należy zapisywać jako `\${`.

Należy pamiętać, iż powyższe rozwiązania są konieczne *wyłącznie* w przypadkach, gdy strona zawiera łańcuch znaków o postaci `${`.

Wyłączanie obsługi języka wyrażeń w całej aplikacji

Język wyrażeń JSP 2.0 jest automatycznie wyłączany w aplikacjach WWW, których deskryptor rozmieszczenia (czyli plik */WEB-INF/web.xml*) odwołuje się do specyfikacji serwletów w wersji 2.3 lub wcześniejszej (czyli JSP 1.2 lub wcześniejszej). Zagadnienia związane z plikami *web.xml* zostały szczegółowo opisane w II tomie książki, jednak krótkie informacje na ich temat można znaleźć w podrozdziale 2.11 („Aplikacje WWW — prezentacja"). Na przykład, poniższy pusty (lecz całkowicie poprawny) plik *web.xml* jest

zgodny ze specyfikacją JSP 1.2, a zatem oznacza, że język wyrażeń powinien zostać domyślnie wyłączony w całej aplikacji.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
</web-app>
```

Z kolei poniższy plik *web.xml* jest zgodny ze specyfikacją JSP 2.0, co sprawia, że język wyrażeń będzie domyślnie włączony. (Oba pliki XML można znaleźć w archiwum przykładów zamieszczonych w niniejszej książce, dostępnym pod adresem *ftp://ftp.helion.pl/przyklady/jsjsp1.zip*).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
           "http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">
</web-app>
```

Wyłączanie obsługi języka wyrażeń na wielu stronach JSP

W aplikacjach WWW, których deskryptor rozmieszczenia informuje, że aplikacja jest tworzona zgodnie ze specyfikacją 2.4 (JSP 2.0), istnieje możliwość określenia grupy stron, na jakich język wyrażeń powinien być ignorowany. Służy do tego element `jsp-property-group` oraz jego atrybut `el-ignored`. Poniższy, przykładowy plik *web.xml* pokazuje, w jaki sposób można wyłączyć obsługę języka wyrażeń we wszystkich stronach JSP umieszczonych w katalogu *stareWersje*:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
           "http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">
  <jsp-property-group>
    <url-pattern>/stareWersje/*.jsp</url-pattern>
    <el-ignored>true</el-ignored>
  </jsp-property-group>
</web-app>
```

Element `jsp-property-group` został szczegółowo opisany w II tomie niniejszej książki.

Wyłączanie obsługi języka wyrażeń na konkretnych stronach JSP

Aby wyłączyć obsługę języka wyrażeń na konkretnej stronie JSP, atrybutowi `isELEnabled` dyrektywy `page` należy przypisać wartość `false`:

```
<%@ page isELEnabled="false" %>
```

Należy zauważyć, iż atrybut `isELEnabled` został wprowadzony w specyfikacji JSP 2.0, a zastosowanie go w aplikacjach zgodnych ze specyfikacją JSP 1.2 zostanie potraktowane jako błąd. Nie można zatem wykorzystać tej metody, by zapewnić poprawne działanie stron JSP na nowszych i starszych serwerach, bez konieczności wprowadzania żadnych zmian w ich kodzie. Dlatego też zastosowanie elementu `jsp-property-group` jest zazwyczaj lepszym rozwiązaniem niż stosowanie atrybutu `isELEnabled`.

Wyłączanie obsługi konkretnych instrukcji języka wyrażeń

Wyobraźmy sobie, że istnieje pewna strona JSP tworzona zgodnie ze specyfikacją JSP 1.2, w której pojawia się łańcuch znaków `${`. Tę stronę należy zastosować w wielu miejscach, w tym także w aplikacjach tworzonych zgodnie ze specyfikacją JSP 1.2, oraz aplikacjach zawierających strony, na których jest stosowany język wyrażeń. Co więcej, musi istnieć możliwość używania tej strony w dowolnych aplikacjach, i to bez wprowadzania *jakichkolwiek* modyfikacji zarówno w kodzie samej strony, jak i w kodzie pliku *web.xml*. Choć taki scenariusz jest raczej mało prawdopodobny, to jednak może się zdarzyć, a żadne z rozwiązań przedstawionych wcześniej nie zapewnia takich możliwości. W takim przypadku wystarczy zamienić znaki `$` występujące w kodzie strony na symbole HTML reprezentujące ten znak. Innymi słowy, każdy umieszczony w kodzie strony łańcuch znaków `${` należy zastąpić łańcuchem `${`. Na przykład, łańcuch znaków:

```
&#36;{uhaha}
```

zostanie zapewne wyświetlony w postaci:

```
${uhaha}
```

Należy jednak pamiętać, iż te symbole HTML są zamieniane na odpowiednie znaki (w tym przypadku na znak `$`) przez *przeglądarkę*, a nie przez *serwer*. A zatem rozwiązanie to można stosować wyłącznie w przypadku generowania kodu HTML, który będzie interpretowany i wyświetlany w przeglądarce WWW.

A co zrobić w przypadku strony JSP tworzonej zgodnie ze specyfikacją JSP 2.0, na której używany jest zarówno język wyrażeń, jak i łańcuchy znaków `${`? W takich sytuacjach przed znakami `$`, które mają być wyświetlone w niezmienionej postaci, należy zapisać znak lewego ukośnika. Na przykład:

```
\${1+1} to ${1+1}
```

zostanie wyświetlone jako:

```
${1+1} to 2
```

16.4. Uniemożliwianie stosowania zwyczajnych elementów skryptowych

Język wyrażeń JSP zapewnia zwarty i przejrzysty sposób dostępu do obiektów Javy przechowywanych w standardowych „miejscach”. Możliwość ta sprawia, że w znacznej mierze można zrezygnować ze stosowania jawnych elementów skryptowych opisanych w rozdziale 11.

W rzeczywistości, niektóry programiści decydują się nawet na całkowitą rezygnację ze stosowania takich standardowych elementów skryptowych. W takim przypadku można użyć elementu `scripting-invalid` umieszczanego wewnątrz elementu `jsp-propety-group`, który sprawia, że zastosowanie standardowego elementu skryptowego zostanie potraktowane jako błąd. Poniższy plik *web.xml* przedstawia przykład zastosowania tego elementu:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
           "http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-invalid>true</scripting-invalid>
  </jsp-property-group>
</web-app>
```

16.5. Odwołania do zmiennych dostępnych w określonym zakresie

W przypadku stosowania architektury MVC (opisanej w rozdziale 15.) serwlet wywołuje kod, który tworzy dane, a następnie przekazuje obsługę żądania do odpowiedniej strony JSP używając do tego celu metod `RequestDispatcher.forward` lub `response.sendRedirect`. Aby strona JSP mogła uzyskać dostęp do tych danych, serwlet musi zapisać je przy użyciu metody `setAttribute` w jednym ze standardowych „miejsc": obiekcie `HttpServletRequest`, obiekcie `HttpSession` lub obiekcie `ServletContext`.

Obiekty zapisane w tych „miejscach" są nazywane „zmiennymi dostępnymi w określonym zakresie" (ang. *scoped variables*), a język wyrażeń JSP 2.0 zapewnia możliwość łatwego i szybkiego dostępu do nich. Takie zmienne można także zapisywać w obiektach `PageContext`, jednak to rozwiązanie jest znacznie mniej użyteczne, gdyż obiekty `PageContext` nie są współużytkowane przez serwlety i strony JSP. A zatem zmienne dostępne w zakresie strony reprezentują jedynie obiekty zapisane wcześniej na tej samej stronie JSP, a nie obiekty zapisane przez serwlet.

Aby wyświetlić zawartość zmiennej dostępnej w pewnym zakresie, należy podać jej nazwę w elemencie języka wyrażeń. Na przykład, wyrażenie:

```
${nazwa}
```

oznacza, że należy przeszukać obiekty `PageContext`, `HttpServletRequest`, `HttpSession` oraz `ServletContext` (dokładnie w podanej kolejności), w poszukiwaniu atrybutu o nazwie *nazwa*. Jeśli uda się odnaleźć atrybut, to wywoływana jest jego metoda `toString`, a uzyskany w ten sposób łańcuch znaków jest zwracany jako wartość wyrażenia. A zatem poniższe dwa wyrażenia zwrócą identyczne wyniki:

```
${nazwa}
<%= pageContext.findAttribute("nazwa") %>
```

Jak widać, drugie z przedstawionych wyrażeń jest znacznie dłuższe, a co gorsza, wymaga jawnego zastosowania kodu napisanego w języku Java. Istnieje co prawda możliwość wyeliminowania kodu Javy, jednak w efekcie trzeba użyć jeszcze bardziej rozbudowanego znacznika akcji `jsp:useBean`:

```
<jsp:useBean id="nazwa" type="jakisPakiet.JakasKlasa" scope="...">
<%= nazwa %>
```

Poza tym, w przypadku stosowania znacznika `jsp:useBean` należy wiedzieć, w jakim zakresie została zapisana zmienna, oraz znać pełną nazwę jej klasy. Stanowi to dosyć duży problem, zwłaszcza jeśli tworzeniem stron JSP nie zajmuje się autor serwletu, lecz inna osoba.

Wybór nazw atrybutów

Przy stosowaniu języka wyrażeń do operowania na zmiennych dostępnych w określonych zakresach, nazwy tworzonych atrybutów muszą być zgodne ze wszystkimi zasadami określającymi dopuszczalną postać nazw zmiennych w języku Java. A zatem nie należy umieszczać w nich kropek, odstępów, ukośników oraz wszelkich innych znaków, które można stosować w zwyczajnych łańcuchach znaków, lecz nie w nazwach zmiennych.

Oprócz tego, należy pamiętać, by nazwy nadawane atrybutom nie kolidowały z nazwami zmiennych predefiniowanych, przedstawionymi w podrozdziale 16.8.

Przykład

W ramach przedstawienia sposobu korzystania ze zmiennych dostępnych w określonym zakresie, serwlet `ScopedVars` (przedstawiony na listingu 16.1) zapisuje pewien łańcuch znaków w obiekcie `HttpServletRequest`, inny łańcuch w obiekcie `HttpSession` oraz datę w obiekcie `ServletContext`. Następnie obsługa żądania jest przekazywana na stronę JSP (przedstawioną na listingu 16.2 oraz rysunku 16.2), na której atrybuty te są wyświetlane przy użyciu wyrażenia `${nazwaAtrybutu}`.

Listing 16.1. *ScopedVars.java*

```
package coreservlets;

/** Serwlet tworzy zmienne dostępne w określonym zakresie
 * (czyli obiekty zapisywane jako atrybuty w pewnych
 * standardowych miejscach). Następnie przekazuje obsługę
 * żądania do strony JSP, która wyświetla wartości zapisane
 * przez serwlet przy użyciu języka wyrażeń.
 */

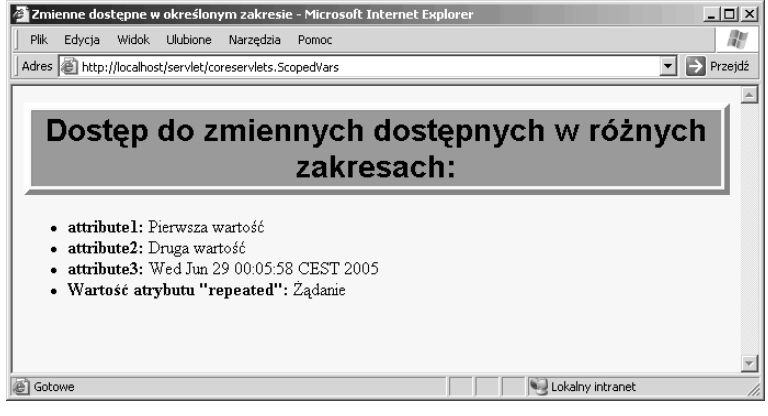
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ScopedVars extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
```

```
    request.setAttribute("attribute1", "Pierwsza wartość");
    HttpSession session = request.getSession();
    session.setAttribute("attribute2", "Druga wartość");
    ServletContext application = getServletContext();
    application.setAttribute("attribute3",
                             new java.util.Date());
    request.setAttribute("repeated", "Żądanie");
    session.setAttribute("repeated", "Sesja");
    application.setAttribute("repeated", "Obiekt ServletContext");
    RequestDispatcher dispatcher =
      request.getRequestDispatcher("/el/scoped-vars.jsp");
    dispatcher.forward(request, response);
  }
}
```

Listing 16.2. *scoped-vars.jsp*

```
<%@ page contentType="text/html; charset=ISO-8859-2" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Zmienne dostępne w określonym zakresie</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-2">
<LINK REL=STYLESHEET
      HREF="/el/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  Dostęp do zmiennych dostępnych w różnych zakresach:
</TABLE>
<P>
<UL>
  <LI><B>attribute1:</B> ${attribute1}
  <LI><B>attribute2:</B> ${attribute2}
  <LI><B>attribute3:</B> ${attribute3}
  <LI><B>Wartość atrybutu "repeated":</B> ${repeated}
</UL>
</BODY></HTML>
```



Rysunek 16.2. *Język wyrażeń JSP 2.0 upraszcza dostęp do zmiennych dostępnych w określonych zakresach — czyli obiektów zapisywanych jako atrybuty obiektu strony, żądania, sesji lub kontekstu serwletu*

Warto zwrócić uwagę, iż strona JSP odwołuje się do wszystkich atrybutów przy użyciu tego samego zapisu, niezależnie od tego, w jakim zakresie zostały one zapisane. Zazwyczaj takie rozwiązanie jest wygodne, gdyż serwlety obsługujące rozwiązania typu MVC nadają zapamiętywanym obiektom unikalne nazwy atrybutów. Niemniej jednak, z technicznego punktu widzenia, istnieje możliwość kilkukrotnego zastosowania tej samej nazwy. Dlatego należy pamiętać, że język wyrażeń poszukuje atrybutu w *następującej* kolejności: obiekt `PageContext`, obiekt `HttpServletRequest`, obiekt `HttpSession` i na końcu obiekt `ServletContext`. Aby przedstawić ten aspekt działania języka wyrażeń, serwlet zapisuje obiekt w trzech różnych zakresach, za każdym razem nadając mu tę samą wartość klucza — `repeated`. Wartość zwrócona przez wyrażenie `${repeated}` (widoczna na rysunku 16.2) odpowiada pierwszemu atrybutowi odnalezionemu podczas przeszukiwania zakresów (w tym przypadku zostanie ona odczytana z obiektu `HttpServletRequest`). Informacje na temat ograniczania poszukiwania atrybutów do konkretnego zakresu zostały podane w podrozdziale 16.8 („Odwołania do obiektów niejawnych").

16.6. Dostęp do właściwości komponentów

W razie zastosowania przedstawionego wcześniej zapisu o postaci `${nazwa}`, system odnajduje obiekt `name`, zmienia go na łańcuch znaków i zwraca. Choć takie działanie jest wygodne, to jednak bardzo rzadko trzeba będzie wyświetlić *wartość samego obiektu* zapisanego przez serwlet. Zazwyczaj będzie chodziło o wyświetlenie *konkretnych właściwości* tego obiektu.

Język wyrażeń JSP udostępnia prostą, a jednocześnie dającą bardzo duże możliwości notację kropkową, służącą właśnie do pobierania wartości właściwości obiektów. Aby pobrać wartość właściwości `firstName` zmiennej o nazwie `klient` dostępnej w pewnym zakresie, wystarczy użyć wyrażenia o postaci `${klient.firstName}`. Choć to wyrażenie wydaje się być bardzo proste, to jednak, aby możliwe było wyznaczenie jego wartości, system musi wykorzystać technologię odzwierciedlania (badania wewnętrznej struktury obiektu). A zatem, zakładając, że obiekt jest typu `NameBean`, który należy do pakietu `coreservlets`, to aby wykonać tą samą czynność przy użyciu zwykłego kodu Javy, należałoby zastąpić wyrażenie:

```
${klient.firstName}
```

następującym blokiem kodu:

```
<% page import="coreservlets.NameBean" %>
<%
  NameBean klient =
             (NameBean) pageContext.findAttribute("klient");
%>
<%= klient.getfirstName %>
```

Co więcej, w przypadku zastosowania języka wyrażeń, jeśli nie uda się odnaleźć atrybutu, to zostanie zwrócony pusty łańcuch znaków. Z kolei stosując standardowe elementy skryptowe, należałyby dodać kolejny fragment kodu Javy, zabezpieczający przez zgłoszeniem wyjątku `NullPointerException`.

Czytelnik zapewne pamięta, że elementy skryptowe JSP nie są jedyną alternatywą dla języka wyrażeń. O ile tylko znany jest zakres, w jakim został zapisany obiekt, oraz pełna nazwa jego klasy, to wyrażenie:

```
${klient.firstName}
```

można zastąpić znacznikami akcji:

```
<jsp:useBean id="klient" type="coreservlets.NameBean"
             scope="request, session lub application" />
<jsp:getProperty name="klient" property="firstName" />
```

Jednak język wyrażeń pozwala na dowolne zagnieżdżanie właściwości. A zatem, gdyby klasa `NameBean` dysponowała właściwością `address` (czyli metodą `getAddress`) zwracającą obiekt typu `Address`, który zawierałby właściwość `zipCode` (czyli metodę `getZipCode`), to do właściwości `zipCode` można by się odwołać w następujący sposób:

```
${klient.address.zipCode}
```

Podobnego odwołania nie można wykonać przy użyciu znaczników `jsp:useBean` oraz `jsp:getProperty` — konieczne byłoby zastosowanie jawnego kodu napisanego w Javie.

Równoważność notacji kropkowej oraz notacji charakterystycznej dla tablic

Warto także zapamiętać, iż język wyrażeń pozwala na zastąpienie notacji kropkowej zapisem charakterystycznym dla odwołań do elementów tablic (wykorzystującym nawiasy kwadratowe). A zatem, zamiast wyrażenia:

```
${nazwa.wlasciwosc}
```

można napisać:

```
${nazwa["wlasciwosc"]}
```

W przypadku odwoływania się do właściwości komponentów, ten drugi sposób zapisu jest stosowany raczej rzadko. Niemniej jednak ma on dwie zalety.

Przede wszystkim, pozwala na wyznaczenie nazwy właściwości w czasie obsługi żądania. Taki sposób zapisu pozwala bowiem, by wartość umieszczona wewnątrz nawiasów sama była zmienną — w przypadku zastosowania notacji kropkowej, musi to być literał.

Poza tym, sposób zapisu charakterystyczny dla tablic pozwala na stosowanie nazw, które jako nazwy właściwości zostałyby uznane za błędne. Możliwość ta nie ma znaczenia podczas operowania na właściwościach komponentów, jednak okazuje się niezwykle przydatna w przypadku odwołań do kolekcji (patrz podrozdział 16.7) lub nagłówków żądania (patrz podrozdział 16.8).

Przykład

Na listingu 16.3 został przedstawiony przykładowy serwlet `BeanProperties`. Serwlet ten tworzy komponent `EmployeeBean` (przedstawiony na listingu 16.4), zapisuje go w obiekcie żądania, jako atrybut o nazwie `employee`, po czym przekazuje żądanie do strony JSP.

Listing 16.3. *BeanProperties.java*

```
package coreservlets;

/** Serwlet tworzący kilka komponentów, których właściwości
 * zostaną wyświetlone przy użyciu języka wyrażeń JSP 2.0.
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class BeanProperties extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    NameBean name = new NameBean("Marty", "Hall");
    CompanyBean company =
      new CompanyBean("coreservlets.com",
                      "J2EE - Kursy i konsulatacje");
    EmployeeBean employee = new EmployeeBean(name, company);
    request.setAttribute("employee", employee);
    RequestDispatcher dispatcher =
      request.getRequestDispatcher("/el/bean-properties.jsp");
    dispatcher.forward(request, response);
  }
}
```

Listing 16.4. *EmployeeBean.java*

```
package coreservlets;

public class EmployeeBean {
  private NameBean name;
  private CompanyBean company;

  public EmployeeBean(NameBean name, CompanyBean company) {
    setName(name);
    setCompany(company);
  }

  public NameBean getName() { return(name); }

  public void setName(NameBean newName) {
    name = newName;
  }

  public CompanyBean getCompany() { return(company); }
```

```
  public void setCompany(CompanyBean newCompany) {
    company = newCompany;
  }
}
```

Klasa `EmployeeBean` definiuje dwie właściwości — `name` oraz `company` — zawierające odpowiednio obiekty typów `NameBean` (patrz listing 16.5) oraz `CompanyBean` (patrz listing 16.6). Klasa `NameBean` definiuje właściwości `lastName` oraz `firstName`, z kolei klasa `CompanyBean` definiuje właściwości `companyName` oraz `business`. Strona JSP, przedstawiona na listingu 16.7, wyświetla wartości wszystkich czterech atrybutów, używając do tego celu następujących wyrażeń EL:

```
${employee.name.firstName}
${employee.name.lastName}
${employee.company.companyName}
${employee.company.business}
```

Listing 16.5. *NameBean.java*

```
package coreservlets;

public class NameBean {
  private String firstName = "Nie podano imienia";
  private String lastName = "Nie podano nazwiska";

  public NameBean() {}

  public NameBean(String firstName, String lastName) {
    setFirstName(firstName);
    setLastName(lastName);
  }

  public String getFirstName() {
    return(firstName);
  }

  public void setFirstName(String newFirstName) {
    firstName = newFirstName;
  }

  public String getLastName() {
    return(lastName);
  }

  public void setLastName(String newLastName) {
    lastName = newLastName;
  }
}
```

Listing 16.6. *CompanyBean.java*

```
package coreservlets;

public class CompanyBean {
  private String companyName;
  private String business;
```

```
public CompanyBean(String companyName, String business) {
    setCompanyName(companyName);
    setBusiness(business);
  }

  public String getCompanyName() { return(companyName); }

  public void setCompanyName(String newCompanyName) {
    companyName = newCompanyName;
  }

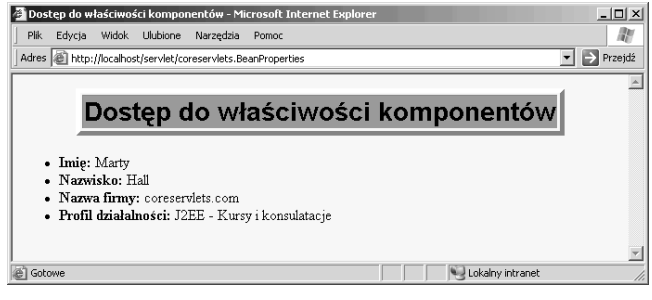
  public String getBusiness() { return(business); }

  public void setBusiness(String newBusiness) {
    business = newBusiness;
  }
}
```

Listing 16.7. *bean-properties.jsp*

```
<%@ page contentType="text/html; charset=ISO-8859-2" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Dostęp do właściwości komponentów</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-2">
<LINK REL=STYLESHEET
      HREF="/el/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  Dostęp do właściwości komponentów
</TABLE>
<P>
<UL>
  <LI><B>Imię:</B> ${employee.name.firstName}
  <LI><B>Nazwisko:</B> ${employee.name.lastName}
  <LI><B>Nazwa firmy:</B> ${employee.company.companyName}
  <LI><B>Profil działalności:</B> ${employee.company.business}
</UL>
</BODY></HTML>
```

Wyniki wykonania strony przedstawiono na rysunku 16.3.



Rysunek 16.3. *Dostęp do właściwości komponentów jest możliwy przy użyciu notacji kropkowej lub zapisu charakterystycznego dla odwołań do tablic*

16.7. Dostęp do zawartości kolekcji

Język wyrażeń JSP 2.0 zapewnia jednolity sposób dostępu do kolekcji różnych typów. Bazuje on na zastosowaniu zapisu charakterystycznego dla odwołań do elementów tablic. Na przykład, jeśli zmienna o nazwie `nazwaAtrybutu` zawiera tablicę, obiekt typu `List` lub `Map`, to do zawartości takiej kolekcji można się odwołać, używając następującego zapisu:

```
${nazwaAtrybutu[nazwaElementu]}
```

Jeśli taka zamienna jest tablicą, to nazwa elementu podawana w wyrażeniu EL będzie indeksem tej tablicy, a wartość całego wyrażenia zostanie pobrana przy użyciu kodu: `tablica[indeks]`. Na przykład, jeśli zmienna `nazwyKlientow` zawiera tablicę łańcuchów znaków, to wyrażenie:

```
${nazwyKlientow[0]}
```

zwróci zawartość pierwszego elementu tej tablicy.

Jeśli zmienna będzie obiektem implementującym interfejs `List`, to nazwa elementu podana w wyrażeniu EL będzie indeksem, a wartość całego wyrażenia zostanie pobrana przy użyciu kodu: `lista.get(indeks)`. Na przykład, jeśli zmienna `nazwyDostawcow` reprezentuje obiekt typu `ArrayList`, to wyrażenie:

```
${nazwyDostawcow[0]}
```

zwróci pierwszy element listy.

Jeśli zmienna będzie obiektem implementującym interfejs `Map`, to nazwa elementu podana w wyrażeniu EL będzie kluczem, a wartość całego wyrażenia zostanie pobrana przy użyciu kodu `mapa.get(klucz)`. Na przykład, zakładając, że zmienna `stoliceStanow` jest obiektem typu `HashMap`, którego klucze określają nazwy stanów USA, a skojarzone z nimi wartości określają nazwy miast będących stolicami tych stanów, to wyrażenie:

```
${stoliceStanow["maryland"]}
```

zwróci łańcuch znaków `"annapolis"`. Jeśli klucze obiektu `Map` spełniają wymagania narzucane dopuszczalnym postaciom nazw zmiennych w języku Java, to zamiast zapisu z nawiasami kwadratowymi można użyć notacji kropkowej. A zatem, powyższe wyrażenie można także zapisać w postaci:

```
${stoliceStanow.maryland}
```

Należy jednak pamiętać, iż zastosowanie zapisu charakterystycznego dla odwołań do tablic pozwala na określanie wartości klucza w czasie obsługi żądania; w przypadku użycia notacji kropkowej wartość klucza musi być znana podczas pisania programu.

Przykład

Zastosowanie wyrażeń EL do pobierania zawartości kolekcji zostało przedstawione na przykładzie serwletu `Collections` (patrz listing 16.8), który tworzy tablicę łańcuchów znaków, obiekt typu `ArrayList` oraz obiekt typu `HashMap`. Następnie serwlet przekazuje żądanie do strony JSP (przedstawionej na listingu 16.9 oraz na rysunku 16.4), która wyświetla elementy tych trzech obiektów używając wyrażeń EL.

Listing 16.8. *Collections.java*

```
package coreservlets;

import java.util.*;

/** Serwlet tworzy kolekcje, których elementy będą wyświetlane
 * przy użyciu języka wyrażeń JSP 2.0.
 */

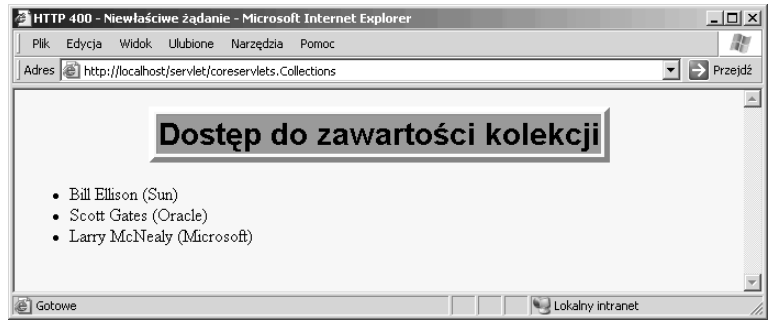
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Collections extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String[] firstNames = { "Bill", "Scott", "Larry" };
    ArrayList lastNames = new ArrayList();
    lastNames.add("Ellison");
    lastNames.add("Gates");
    lastNames.add("McNealy");
    HashMap companyNames = new HashMap();
    companyNames.put("Ellison", "Sun");
    companyNames.put("Gates", "Oracle");
    companyNames.put("McNealy", "Microsoft");
    request.setAttribute("first", firstNames);
    request.setAttribute("last", lastNames);
    request.setAttribute("company", companyNames);
    RequestDispatcher dispatcher =
      request.getRequestDispatcher("/el/collections.jsp");
    dispatcher.forward(request, response);
  }
}
```

Listing 16.9. *collections.jsp*

```
<%@ page contentType="text/html; charset=ISO-8859-2" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Dostęp do zawartości kolekcji</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-2">
<LINK REL=STYLESHEET
      HREF="/el/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  Dostęp do zawartości kolekcji
  </TH></TR>
</TABLE>
<P>
<UL>
  <LI>${first[0]} ${last[0]} (${company["Ellison"]})
  <LI>${first[1]} ${last[1]} (${company["Gates"]})
```

```
  <LI>${first[2]} ${last[2]} (${company["McNealy"]})
</UL>
</BODY></HTML>
```



Rysunek 16.4. *Podczas odwoływania się do elementów kolekcji można używać notacji kropkowej lub zapisu wykorzystującego nawiasy kwadratowe. Notacja kropkowa może być używana wyłącznie w przypadku, gdy nazwy kluczy spełniają wymagania określające dopuszczalną postać nazw zmiennych w Javie*

Nazwy właściwości komponentów nie mogą składać się wyłącznie z cyfr, dlatego podczas odwoływania się do elementów tablic oraz zawartości obiektów `ArrayList` należy używać zapisu wykorzystującego nawiasy kwadratowe. Jednak zawartość obiektów `HashMap` można pobierać, używając zarówno zapisu z nawiasami kwadratowymi, jak i notacji kropkowej. W celu uzyskania spójnego zapisu wyrażeń, w stronie JSP przedstawionej na listingu 16.9 zastosowano zapis wykorzystujący nawiasy kwadratowe, jednak użyte na stronie wyrażenie:

```
${company["Ellison"]}
```

z powodzeniem można by zastąpić wyrażeniem o postaci:

```
${company.Ellison}
```

16.8. Odwołania do obiektów niejawnych

W większości przypadków język wyrażeń JSP jest stosowany w aplikacjach działających według zasad architektury MVC (model-widok-kontroler, opisanej w rozdziale 15.). W aplikacjach tego typu serwlet, do którego są kierowane żądania, tworzy niezbędne dane, po czym przekazuje obsługę żądania do strony JSP. Zazwyczaj taka strona JSP używa wyłącznie obiektów utworzonych przez serwlet, więc w zupełności wystarczają jej ogólne mechanizmy dostępu do właściwości komponentów oraz elementów kolekcji.

Nie oznacza to wcale, iż języka wyrażeń JSP 2.0 można używać wyłącznie w aplikacjach tego typu — jeśli tylko serwer obsługuje specyfikację JSP 2.0, a plik *web.xml* odwołuje się do specyfikacji JavaServlet 2.4, to język wyrażeń będzie można stosować na wszystkich stronach JSP. Jednak, aby możliwości te były przydatne, w specyfikacji JSP zdefiniowano kilka opisanych poniżej obiektów niejawnych.

pageContext

Obiekt `pageContext` reprezentuje obiekt `PageContext` bieżącej strony JSP. Z kolei klasa `PageContext` definiuje właściwości `request`, `response`, `session`, `out` oraz `servletContext` (czyli metody: `getRequest`, `getResponse`, `getSession`, `getOut` oraz `getServletContext`). A zatem poniższe wyrażenie wyświetla identyfikator bieżącej sesji:

```
${pageContext.session.id}
```

param oraz paramValues

Te dwa obiekty pozwalają odpowiednio na pobranie wartości głównego parametru żądania (`param`) oraz tablicy zawierającej wartości wszystkich parametrów żądania (`paramValues`). Poniższe wyrażenie zwraca wartość parametru żądania `custID` (przy czym, jeśli parametr ten nie zostanie podany, zwracany jest pusty łańcuch znaków, a nie wartość `null`):

```
${param.custID}
```

Więcej informacji na temat parametrów żądania można znaleźć w rozdziale 4.

header oraz headerValues

Te obiekty zapewniają odpowiednio dostęp do głównych oraz wszystkich wartości nagłówków żądania HTTP. Należy pamiętać, iż stosowanie notacji kropkowej nie jest dozwolone, jeśli umieszczana po kropce nazwa nie jest poprawną nazwą właściwości. A zatem wartość nagłówka `Accept` można pobrać na dwa sposoby:

```
${header.Accept}
```

oraz

```
${header["Accept"]}
```

Z kolei wartość nagłówka `Accept-Encoding` można pobrać wyłącznie przy użyciu wyrażenia o postaci:

```
${header["Accept-Encoding"]}
```

Więcej informacji na temat nagłówków żądania HTTP można znaleźć w rozdziale 5.

cookie

Obiekt `cookie` pozwala na szybki dostęp do cookies przesyłanych z przeglądarki. Jednak koniecznie należy pamiętać, że zwracaną wartością jest obiekt `Cookie`, a nie sama wartość cookie. A zatem w celu odczytania wartości cookie, należy dodatkowo użyć właściwości `value` (czyli metody `getValue`) obiektu `Cookie`. Na przykład, oba poniższe wyrażenia wyświetlają wartość cookie o nazwie `cookieUzytkownika` (lub pusty łańcuch znaków, jeśli wskazane cookie nie zostanie odnalezione):

```
${cookie.cookieUzytkownika.value}
${cookie["cookieUzytkownika"].value}
```

Więcej informacji na temat cookies można znaleźć w rozdziale 8.

initParam

Obiekt `initParam` pozwala na bardzo łatwy dostęp do parametrów inicjalizacyjnych kontekstu. Na przykład, poniższe wyrażenie wyświetla wartość parametru inicjalizacyjnego o nazwie `domyslnyKolor`:

```
${initParam.domyslnyKolor}
```

Więcej informacji dotyczących parametrów inicjalizacyjnych można znaleźć w II tomie niniejszej książki.

pageScope, requestScope, sessionScope oraz applicationScope

Przy użyciu tych czterech obiektów można ograniczyć „miejsca", w jakich system będzie poszukiwać zmiennych dostępnych w określonych zakresach. Na przykład, w razie użycia wyrażenia:

```
${nazwa}
```

system poszukuje zmiennej o nazwie `nazwa` najpierw w obiekcie `PageContext`, następnie `HttpServletRequest`, `HttpSession`, i w końcu w obiekcie `ServletContext`; zwrócona zostanie pierwsza odnaleziona zmienna. Jednak w razie użycia następującego wyrażenia:

```
${requestScope.nazwa}
```

system będzie poszukiwać zmiennej wyłącznie w obiekcie `HttpServletRequest`.

Przykład

Strona przedstawiona na listingu 16.10 używa obiektów niejawnych w celu wyświetlenia parametrów żądania, nagłówków żądania HTTP, wartości `cookies` oraz informacji dotyczących serwera. Wyniki generowane przez tę stronę zostały przedstawione na rysunku 16.5.

Listing 16.10. *implicit-objects.jsp*

```
<%@ page contentType="text/html; charset=ISO-8859-2" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Using Implicit Objects</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-2">
<LINK REL=STYLESHEET
      HREF="/el/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  Zastosowanie obiektów niejawnych
  </TH></TR>
</TABLE>
<P>
<UL>
  <LI><B>Parametr żądania "test":</B> ${param.test}
  <LI><B>Nagłówek User-Agent:</B> ${header["User-Agent"]}
```

```
  <LI><B>Wartość cookie "JSESSIONID":</B> ${cookie.JSESSIONID.value}
  <LI><B>Serwer:</B> ${pageContext.servletContext.serverInfo}
</UL>
</BODY></HTML>
```



Rysunek 16.5. *Dostępnych jest kilka zmiennych definiowanych automatycznie. Takie predefiniowane zmienne są nazywane „obiektami niejawnymi”*

16.9. Stosowanie operatorów języka wyrażeń

Język wyrażeń JSP 2.0 definiuje kilka grup operatorów; należą do nich operatory: arytmetyczne, relacyjne, logiczne oraz operator sprawdzający, czy została określona wartość zmiennej. Choć stosując te operatory można tworzyć wyrażenia nieco prostsze od równoważnego kodu Javy, to jednak nie należy zapominać o podstawowym celu, w jakim został opracowany język wyrażeń: zapewnieniu zwięzłego sposobu dostępu do istniejących obiektów i to zazwyczaj w aplikacjach działających według zasad architektury MVC. Trudno zatem podać poważne zalety zastępowania długich elementów skryptowych pisanych w Javie przez niewiele krótsze wyrażenia EL. Oba te rozwiązania są z założenia błędne — złożona logika biznesowa lub kod służący do pobierania informacji w ogóle nie powinien być umieszczany na stronach JSP. W możliwie jak największym stopniu zastosowanie języka wyrażeń należy ograniczać do logiki *prezentacji*; logika biznesowa powinna być implementowana w normalnych klasach Javy i wywoływana przez serwlety.

> Operatory języka wyrażeń powinny być używane w prostych zastosowaniach, które mają bezpośredni związek z logiką prezentacji (na przykład, przy podjęciu decyzji, w jaki sposób należy wyświetlić dane). Należy unikać stosowania tych operatorów w logice biznesowej (czyli do tworzenia i przetwarzania danych). Logikę biznesową należy implementować w zwyczajnych klasach Javy, używanych w kodzie serwletów, które, w architekturze MVC, rozpoczynają proces obsługi żądań.

Operatory arytmetyczne

Operatory arytmetyczne dostępne w języku wyrażeń są zazwyczaj używane do wykonywania prostych operacji na wartościach zapisanych w komponentach. Nie należy stosować ich do realizacji złożonych algorytmów; taki kod należy umieszczać w zwyczajnych klasach Javy.

Operatory + oraz –

To standardowe operatory dodawania i odejmowania, które mają dwie szczególne cechy. Po pierwsze, jeśli którykolwiek z operandów jest łańcuchem znaków, to przed wykonaniem działania zostanie on skonwertowany do postaci liczby (trzeba przy tym pamiętać, że system nie przechwytuje automatycznie wyjątków `NumberFormatException`). Poza tym, jeśli którykolwiek z operandów jest typu `BigInteger` lub `BigDecimal`, to system wykona działania używając odpowiednich metod `add` lub `subtract`.

Operatory *, / oraz div

To standardowe operatory mnożenia i dzielenia, które mają kilka cech szczególnych. Przede wszystkim, podobnie jak w przypadku operatorów + oraz -, typy operandów są konwertowane automatycznie. Po drugie, podczas wykonywania działań uwzględniany jest standardowy priorytet operatorów. A zatem poniższe wyrażenie:

```
${1 + 2 * 3 }
```

zwróci wartość 7, a nie 9. Kolejność wykonywania poszczególnych działań można zmieniać przy użyciu nawiasów. Po trzecie, operatory `/` oraz `div` mają to samo działanie; są one dostępne w celu zapewnienia zgodności z językami XPath oraz JavaScript (ECMAScript).

Operatory % oraz mod

Operator `%` (oraz stanowiący jego odpowiednik operator `mod`) obliczają moduł (czyli resztę z dzielenia) analogicznie jak operator `%` języka Java.

Operatory relacyjne

Operatory relacyjne najczęściej są stosowane wraz z operatorem warunkowym udostępnianym przez język wyrażeń (patrz podrozdział 16.10), bądź też w znacznikach niestandardowych, których atrybuty wymagają podania wartości logicznej (na przykład, w znacznikach umożliwiających wielokrotne przetwarzanie, takich jak te dostępne w bibliotece JSLT, opisanej w II tomie niniejszej książki).

Operatory == oraz eq

Te dwa równoważne sobie operatory sprawdzają, czy podane argumenty są sobie równe. Jednak pod względem sposobu działania przypominają one raczej metodę `equals` Javy, niż stosowany w tym języku operator ==. Operatory te zwracają wartość `true`, jeśli oba operandy są tym samym obiektem. Jeśli operandy są liczbami, to zostają one porównane przy użyciu standardowego operatora == Javy. Jeśli jeden z operandów ma wartość `null`, to porównanie zawsze zwraca wartość `false`. Jeśli któryś z operandów jest obiektem typu `BigInteger` lub `BigDecimal`, to operandy są porównywane przy użyciu metody `compareTo`. We wszystkich pozostałych przypadkach operandy są porównywane przy użyciu metody `equals`.

Operatory != oraz ne

Te dwa równoważne sobie operatory sprawdzają, czy podane argumenty są różne od siebie. Jednak także i one pod względem sposobu działania przypominają raczej negację metody `equals` Javy, niż stosowany w tym języku operator `!=`. Jeśli oba operandy są tym samym obiektem, to operatory te zwracają wartość `false`. Jeśli operandy są liczbami, to zostają one porównane przy użyciu standardowego operatora `!=` Javy. Jeśli jeden z operandów ma wartość `null`, to porównanie zawsze zwraca wartość `true`. Jeśli któryś z operandów jest obiektem typu `BigInteger` lub `BigDecimal`, to operandy są porównywane przy użyciu metody `compareTo`. We wszystkich pozostałych przypadkach operandy są porównywane przy użyciu metody `equals`, a wynik wyrażenia jest negacją wartości zwróconej przez tę metodę.

Operatory < i lt, > i gt, <= i le oraz >= i ge

Są to standardowe operatory relacyjne, które mają jednak dwie cechy szczególne. Przede wszystkim, porównywane operandy są poddawane konwersji typów (analogicznie jak w przypadku operatorów == oraz `!=`). A poza tym, jeśli operandy są łańcuchami znaków, to zostaną one porównane alfabetycznie.

Operatory logiczne

Te operatory służą do łączenia wyników zwracanych przez operatory relacyjne.

Operatory &&, and, ||, or, ! oraz not

To standardowe operatory logicznej koniunkcji (AND), alternatywy (OR) oraz zaprzeczenia (!). W pierwszej kolejności konwertują one operandy do wartości logicznych, a następnie używają standardowych operatorów logicznych Javy o „skróconym cyklu przetwarzania" (które sprawiają, że przetwarzanie wyrażenia zostaje przerwane, kiedy tylko zostanie określony jego ostateczny wynik). Operator `&&` jest równoważny z operatorem `and`, operator `||` jest równoważny z `or`, a `!` — równoważny z `not`.

Operator empty

Ten operator zwraca wartość `true`, jeśli jego argumentem jest wartość `null`, pusty łańcuch znaków, pusta tablica, pusty obiekt `Map` lub pusta kolekcja. W przeciwnym razie operator zwraca wartość `false`.

Przykład

Listing 16.11 przedstawia zastosowanie kilku standardowych operatorów. Wyniki zostały pokazane na rysunku 16.6.

Listing 16.11. *operators.jsp*

```
<%@ page contentType="text/html; charset=ISO-8859-2" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Operatory języka wyrażeń</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-2">
<LINK REL=STYLESHEET
      HREF="/el/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  Operatory języka wyrażeń
  </TH></TR>
</TABLE>
<P>
<TABLE BORDER=1 ALIGN="CENTER">
  <TR><TH CLASS="COLORED" COLSPAN=2>Operatory arytmetyczne</TH>
      <TH CLASS="COLORED" COLSPAN=2>Operatory relacyjne</TH>
  </TR>
  <TR><TH>Wyrażenie</TH><TH>Wynik</TH><TH>Wyrażenie</TH><TH>Wynik</TH></TR>
  <TR ALIGN="CENTER">
    <TD>\${3+2-1}<TD>${3+2-1}  <%-- Dodawanie/odejmowanie --%></TD>
    <TD>\${1&lt;2}<TD>${1<2}      <%-- Porównanie liczb --%></TD>
  </TR>
  <TR ALIGN="CENTER">
    <TD>\${"1"+2}<TD>${"1"+2}     <%-- Konwersja łańcuchów znaków --%></TD>
    <TD>\${"a"&lt;"b"}<TD>${"a"<"b"} <%-- Porównanie alfabetyczne --%></TD>
  </TR>
  <TR ALIGN="CENTER">
    <TD>\${1 + 2*3 + 3/4}<TD>${1 + 2*3 + 3/4}  <%-- Mnożenie/dzielenie --%></TD>
    <TD>\${2/3 &gt;= 3/2}<TD>${2/3 >= 3/2}        <%-- >= --%></TD>
  </TR>
  <TR ALIGN="CENTER">
    <TD>\${3%2}<TD>${3%2}                 <%-- Reszta z dzielenia --%></TD>
    <TD>\${3/4 == 0.75}<TD>${3/4 == 0.75} <%-- Liczbowo = --%></TD>
  </TR>
  <TR ALIGN="CENTER">
    <%-- Operatory div oraz mod są odpowiednikami operatorów / and % --%></TD>
    <TD>\${(8 div 2) mod 3}<TD>${(8 div 2) mod 3}
    <%-- Porównanie przy użyciu "equals" jednak zwraca wartość
     false w razie podania wartości null --%></TD>
    <TD>\${null == "test"}<TD>${null == "test"}</TD>
  </TR>

  <TR><TH CLASS="COLORED" COLSPAN=2>Operatry logiczne</TH>
      <TH CLASS="COLORED" COLSPAN=2> Operator <CODE>empty</CODE></TH>
  </TR>
  <TR><TH>Wyrażenie</TH><TH>Wynik</TH><TH>Wyrażenie</TH><TH>Wynik</TH></TR>
  <TR ALIGN="CENTER">
    <TD>\${(1&lt;2) && (4&lt;3)}<TD>${(1<2) && (4<3)} <%--AND--%></TD>
    <TD>\${empty ""}<TD>${empty ""} <%-- Pusty łańcuch znaków --%></TD>
  </TR>
  <TR ALIGN="CENTER">
    <TD>\${(1&lt;2) || (4&lt;3)}<TD>${(1<2) || (4<3)} <%--OR--%></TD>
    <TD>\${empty null}<TD>${empty null} <%-- null --%></TD>
```

```
  </TR>
  <TR ALIGN="CENTER">
    <TD>\${!(1&lt;2)}<TD>${!(1<2)}  <%-- NOT -%>
    <%-- Obsługa wartości null oraz pustych łańcuchów znaków
     w parametrach żądania --%></TD>
    <TD>\${empty param.blah}<TD>${empty param.blah}</TD>
  </TR>
</TABLE>
</BODY></HTML>
```



Rysunek 16.6. *Język wyrażeń JSP 2.0 udostępnia niewielki zbiór operatorów. Należy ich używać z dużym umiarem — logika biznesowa powinna być wykonywana przez serwlety, a nie na stronach JSP*

16.10. Warunkowe przetwarzanie wyrażeń

Język wyrażeń JSP 2.0 sam w sobie nie udostępnia bogatych możliwości realizacji logiki warunkowej. Możliwości, jakimi dysponuje, w rzeczywistości zapewniają znaczniki `c:if` oraz `c:choose` biblioteki JSTL (ang. *JSP Standard Template Library*), bądź jakiejkolwiek innej biblioteki znaczników niestandardowych. (Zagadnienia związane ze stosowaniem biblioteki JSTL oraz tworzeniem znaczników niestandardowych zostały opisane w II tomie niniejszej książki).

Niemniej jednak, język wyrażeń udostępnia podstawowy operator `?:`, doskonale znany z języków Java, C oraz C++. Na przykład, jeśli wyrażenie `test` przyjmie wartość `true`, to poniższe wyrażenie:

```
${ test ? wyrażenie1 : wyrażenie2 }
```

zwróci wartość wyrażenia `wyrażenie1`; w przeciwnym przypadku zwróci ono wartość wyrażenia `wyrażenie2`. Nie można jednak zapominać, iż podstawowym przeznaczeniem języka wyrażeń jest upraszczanie logiki prezentacji; dlatego należy pamiętać, by nie stosować tego operatora do wykonywania operacji związanych z logiką biznesową.

Przykład

Serwlet przedstawiony na listingu 16.12 tworzy dwa obiekty `SalesBean` (patrz listing 16.13), po czym przekazuje żądanie na stronę JSP (patrz listing 16.14) odpowiedzialną za przedstawienie wyników (patrz rysunek 16.7). Jednak jeśli całkowita ilość sprzedanych produktów będzie mniejsza od 0, to strona JSP powinna zastosować czerwone tło w odpowiedniej komórce tabeli. Implementacja takiego sposobu działania strony należy do zadań logiki prezentacji, a zatem zastosowanie operatora `?:` jest właściwe.

Listing 16.12. *Conditionals.java*

```
package coreservlets;

/** Serwlet tworzy zmienne dostępne w zakresie żądania, które
 *  zostaną zastosowane do przedstawienia operatora warunkowego
 *  języka wyrażeń (xxx ? xxx : xxx).
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Conditionals extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    SalesBean apples =
      new SalesBean(150.25, -75.25, 22.25, -33.57);
    SalesBean oranges =
      new SalesBean(-220.25, -49.57, 138.25, 12.25);
    request.setAttribute("apples", apples);
    request.setAttribute("oranges", oranges);
    RequestDispatcher dispatcher =
      request.getRequestDispatcher("/el/conditionals.jsp");
    dispatcher.forward(request, response);
  }
}
```

Listing 16.13. *SalesBean.java*

```
package coreservlets;

public class SalesBean {
  private double q1, q2, q3, q4;

  public SalesBean(double q1Sales,
                   double q2Sales,
                   double q3Sales,
                   double q4Sales) {
```

```
    q1 = q1Sales;
    q2 = q2Sales;
    q3 = q3Sales;
    q4 = q4Sales;
  }

  public double getQ1() { return(q1); }

  public double getQ2() { return(q2); }

  public double getQ3() { return(q3); }

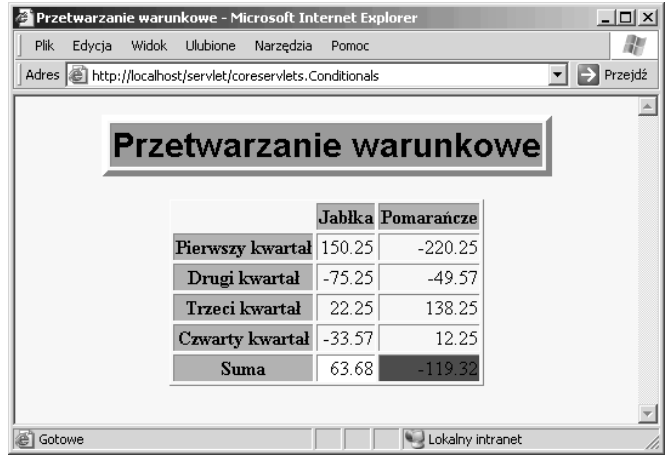
  public double getQ4() { return(q4); }

  public double getTotal() { return(q1 + q2 + q3 + q4); }
}
```

Listing 16.14. *conditionals.jsp*

```
<%@ page contentType="text/html; charset=ISO-8859-2" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Przetwarzanie warunkowe</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-2">
<LINK REL=STYLESHEET
      HREF="/el/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  Przetwarzanie warunkowe
  </TH></TR>
</TABLE>
<P>
<TABLE BORDER=1 ALIGN="CENTER">
  <TR><TH></TH>
      <TH CLASS="COLORED">Jabłka</TH>
      <TH CLASS="COLORED">Pomarańcze</TH>
  </TR>
  <TR><TH CLASS="COLORED">Pierwszy kwartał</TH>
      <TD ALIGN="RIGHT">${apples.q1}</TD>
      <TD ALIGN="RIGHT">${oranges.q1}</TD>
  </TR>
  <TR><TH CLASS="COLORED">Drugi kwartał</TH>
      <TD ALIGN="RIGHT">${apples.q2}</TD>
      <TD ALIGN="RIGHT">${oranges.q2}</TD>
  </TR>
  <TR><TH CLASS="COLORED">Trzeci kwartał</TH>
      <TD ALIGN="RIGHT">${apples.q3}</TD>
      <TD ALIGN="RIGHT">${oranges.q3}</TD>
  </TR>
  <TR><TH CLASS="COLORED">Czwarty kwartał</TH>
      <TD ALIGN="RIGHT">${apples.q4}</TD>
      <TD ALIGN="RIGHT">${oranges.q4}</TD>
  </TR>
  <TR><TH CLASS="COLORED">Suma</TH>
```

```
      <TD ALIGN="RIGHT"
          BGCOLOR="${(apples.total < 0) ? "RED" : "WHITE" }">
      ${apples.total}</TD>
      <TD ALIGN="RIGHT"
          BGCOLOR="${(oranges.total < 0) ? "RED" : "WHITE" }">
      ${oranges.total}</TD>
  </TR>
</TABLE>
</P>
</BODY></HTML>
```



Rysunek 16.7. *W celu warunkowego wyświetlania różnych elementów można zastosować operator ?: przypominający analogiczny operator znany z języka C. Jednak często zdarza się, iż taką warunkową prezentację znacznie wygodniej można zaimplementować przy użyciu niestandardowych znaczników dostępnych w bibliotece znaczników JSTL*

16.11. Informacje o innych możliwościach języka wyrażeń

W niniejszym rozdziale opisano większość możliwości, jakie zapewnia język wyrażeń JSP 2.0. Pominięte zostały jedynie dwie z nich: zastosowanie języka wyrażeń w bibliotekach znaczników niestandardowych oraz zastosowanie funkcji języka wyrażeń. Oba te zagadnienia zostały szczegółowo przedstawione w II tomie niniejszej książki, gdyż dopiero w nim zostały opisane niezbędne technologie.

Oto bardzo skrótowe przedstawienie tych możliwości: wyrażenia EL mogą być umieszczane w dowolnych atrybutach znaczników niestandardowych, które mogą być przetwarzane w trakcie obsługi żądania (czyli atrybutów opisywanych przez element `attribute`, którego atrybut `rtexprvalue` przyjmuje wartość `true`). Z kolei funkcje języka wyrażeń odpowiadają publicznym, statycznym metodom zwyczajnych klas Javy, które w pliku deskryptora biblioteki znaczników (TLD) zostały zdefiniowane przy użyciu elementu `function`.