APLIKACJE WWW W JĘZYKU JAVA!

Apress

Java
Projektowanie aplikacji WWW

Vishal Layka

Helion

Tytuł oryginału: Learn Java for Web Development

Tłumaczenie: Lech Lachowski

ISBN: 978-83-246-9806-6

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/javaps
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 3

Najlepsze praktyki projektowania stron WWW na platformie Java EE

Cała znana ewolucja polega na przechodzeniu od stanu niesprecyzowania do określoności.

— Charles Sanders Peirce

Dobre rozwiązania wymyślane są od czasu do czasu. Czasami są one odkrywane. Wynalazek i odkrycie nie są synonimami[1] i oznaczają one różne cele, choć oba są realizowane poprzez doświadczenie i wiedzę. Doświadczenie pomaga znajdować dobre rozwiązania, które po zastosowaniu dla tych samych zestawów problemów powodują wyłanianie się wzorców. **Wzorce** (ang. *patterns*) są katalogiem dobrych rozwiązań, które wynikają z doświadczenia i specjalistycznej wiedzy programistów.

Architekt Christopher Alexander zauważył, że architekci mają tendencję do rozwiązywania tych samych problemów w mniej więcej taki sam sposób. To spostrzeżenie doprowadziło go do napisania książki o wzorcach projektowych dla architektów[2]. Napisał w tej książce: „Wzorzec projektowy opisuje problem, który się powtarza, a następnie opisuje sedno rozwiązania tego problemu, w taki sposób, że można zastosować je ponad milion razy, bez robienia tego nawet dwukrotnie tak samo".

Pomysły Alexandra w kontekście oprogramowania zastosowano w 1994 r., w oryginalnym wydaniu książki *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku* (wydanie polskie: Helion, 2010) Ericha Gammy, Richarda Helma, Ralpha Johnsona oraz Johna Vlissidesa, znanych jako „Banda Czworga" lub „GoF" (ang. *Gang of Four*). Ta książka opisała wzorce projektowania obiektowego (ang. *object-oriented* — OO) oraz zapoczątkowała falę tworzenia najlepszych rozwiązań praktycznych. Wprowadzając strategię projektowania, którą można wykorzystywać w różnych aplikacjach, odniosła się również do szeregu wymagań projektowych nagromadzonych na przestrzeni lat.

Wraz z rozwojem systemów serwerowych wyłoniła się architektura typu enterprise, taka jak platforma Java EE, która zapewniła abstrakcję technologii i usług. Korzystanie z platformy Java EE nie gwarantuje jednak stosowania najlepszych praktyk architektonicznych i projektowych. John Crupi, Dan Malks i Deepak Alur jako pierwsi ustalili wzorce projektowe dla platformy Java EE na podstawie własnych doświadczeń przy projektowaniu systemów typu enterprise.

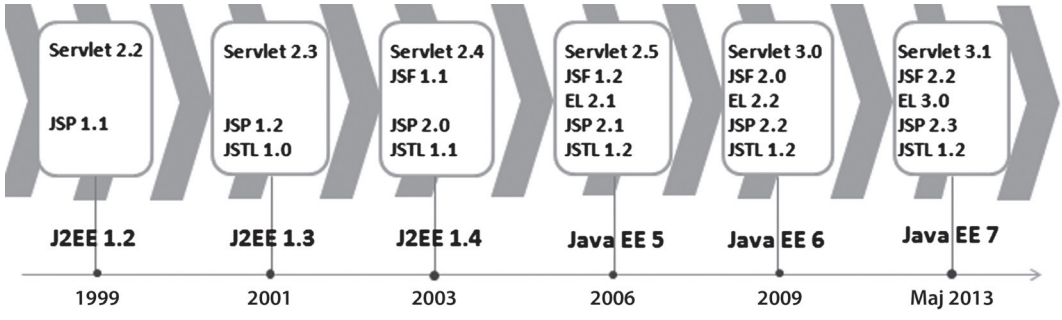
---

[1] **Wynalazek** oznacza tworzenie nowego obiektu z wcześniej istniejących obiektów. **Odkrycie** jest dowiedzeniem się czegoś, co istniało wcześniej, ale bez tworzenia nowych obiektów.

[2] *Język wzorców. Miasta — budynki — konstrukcja* Christophera Alexandra, Sary Ishikawy oraz Murraya Silversteina (Wydawnictwo Gdańskie Psychologiczne, 2009). Zobacz także *The Timeless Way of Building* Christophera Alexandra (Oxford University Press, 1979).

Zastosowanie tych wzorców projektowych do rozwoju aplikacji opartych na platformie Java EE jest **zwyczajowo wymagane** do osiągnięcia najlepszych praktyk architektonicznych i projektowych. Zapewnienie stosowania najlepszych praktyk tworzenia architektury i projektowania nie wystarczy jednak, aby zagwarantować wielokrotną używalność, utrzymywalność oraz rozszerzalność oprogramowania. Nawet z wdrożonymi wzorcami projektowymi platformy Java EE projekt oprogramowania jest często dotknięty przez zjawisko zwane **entropią**, czyli miarą stopnia nieuporządkowania układu. Według drugiego prawa termodynamiki wszystko we wszechświecie przechodzi od niskiego poziomu entropii (stan uporządkowania) do wysokiego poziomu entropii (stan nieuporządkowania) i ewentualnego chaosu. Wszystko w naturze jest związane tym nieodwołalnym prawem fizyki, z którym natura radzi sobie na drodze ewolucji.

Projekt oprogramowania zbyt często ma tendencję, aby przechodzić z niskiego poziomu entropii do wysokiego poziomu entropii. Aby poradzić sobie z tym stanem nieuporządkowania, niezbędna jest ewolucja. To nie znaczy, że rozwijanie projektu zastępuje zarządzanie projektem i metodologię. Pomimo czynników takich jak najbardziej dopracowane zarządzanie projektami, właściwe i wnikliwe metodologie oraz zespół programistów z żywym poczuciem stylu projekt oprogramowania może wpaść w stan wysokiej entropii. Na rysunku 3.1 przedstawiono fazy ewolucji poziomu sieciowego platformy Java EE, które pomagają radzić sobie z entropią w projektach informatycznych.



Rysunek 3.1. *Ewolucja poziomu sieciowego platformy Java EE*

Jak widać na rysunku 3.1, w poziomie sieciowym platformy Java EE nie tylko zachodziła ewolucja istniejących technologii, ale także w każdej nowej wersji dodawane były kolejne technologie. Jeśli nadal używasz platformy J2EE w wersji 1.4, to np. nie będziesz mieć ujednoliconego języka wyrażeń EL (język wyrażeń EL w wersji 2.1) w swoim zestawie narzędziowym. Istotne jest, aby nadążać za rozwojem wszelkich technologii, żeby nie popaść w ewentualny stan wysokiej entropii, która może dotknąć każdy projekt.

Ten rozdział wyjaśnia znaczenie stopniowego rozwijania projektu z nowymi technologiami dostępnymi w każdej nowej wersji platformy Java EE i omawia wykorzystanie ich do poradzenia sobie z wysokim poziomem entropii w projekcie. W dalszej części rozdziału opisano znaczenie wzorców Java EE i wyjaśniono, w jaki sposób za pomocą wzorców poziomu sieciowego Java EE można sprawić, aby aplikacje sieciowe nadawały się do wielokrotnego użytku, były utrzymywalne i rozszerzalne. W kolejnych fragmentach przedstawiono, jak framework WWW pozwoli Ci zrezygnować z używania wzorców poziomu sieciowego Java EE poprzez zapewnienie gotowych najlepszych rozwiązań praktycznych.

Najlepsze rozwiązania praktyczne: korzystanie z języka wyrażeń EL i biblioteki JSTL

Firma Sun opublikowała specyfikację serwletu w 1998 roku. Jedynym celem tworzenia serwletów była pomoc serwerom WWW Java w generowaniu dynamicznej zawartości dla klienta. W listingu 3.1 przedstawiono, jak wyglądał pierwszy serwlet.

Listing 3.1. *Pierwszy serwlet*

```
1.import java.io.IOException;
2.import java.io.PrintWriter;
3.
4.import javax.servlet.ServletException;
5.import javax.servlet.http.HttpServlet;
6.import javax.servlet.http.HttpServletRequest;
7.import javax.servlet.http.HttpServletResponse;
8.public class Hello extends HttpServlet {
9.public void doGet(HttpServletRequest req, HttpServletResponse res)
10.throws ServletException, IOException {
11.res.setContentType ("text/html;charset=ISO-8859-2");
12.PrintWriter out = res.getWriter();
13.out.println("<HTML>");
14.out.println("<HEAD><TITLE>Witaj, świecie!</TITLE></HEAD>");
15.out.println("<BODY>");
16.out.println("<BIG>Witaj, świecie!</BIG>");
17.out.println("</BODY></HTML>");
18.}
19.}
```

Serwlety sprawdzały się świetnie przy generowaniu dynamicznej treści, ale miały jeden poważny problem. Widok był ściśle związany z serwletem, jak widać w wierszach 13. – 17. w listingu 3.1. Aby rozwiązać ten problem, stworzono technologię JSP, która usuwa potrzebę ścisłego związania kodu widoku z kodem logiki biznesowej. To rozdzielenie zagadnień widoku od zagadnień logiki biznesowej zależy od atrybutów podanych w listingach 3.2 i 3.3.

Listing 3.2. *Używanie atrybutów do rozdzielenia kodu prezentacji (widok) od kodu biznesowego (serwlet)*

```
1.public void doPost(HttpServletRequest request, HttpServletResponse response)
2.throws IOException, ServletException {
3.String name = request.getParameter("nazwa_użytkownika");
4.request.setAttribute("name", name);
5.RequestDispatcher view = request.getRequestDispatcher("/result.jsp");
6.view.forward(request, response);
7.}
```

Listing 3.2 przedstawia fragment serwletu używającego atrybutów.

- **Wiersz 3.** — pobiera wartość *nazwa_użytkownika* z żądania.
- **Wiersz 4.** — ustawia wartość *nazwa_użytkownika* jako atrybut w żądaniu.
- **Wiersz 5.** — pobiera interfejs `RequestDispatcher` z żądania.
- **Wiersz 6.** — przekazuje do widoku żądanie i obiekt odpowiedzi. Należy zwrócić uwagę, że obiekt żądania posiada ustawiony atrybut dla wartości *nazwa_użytkownika*. Widok może teraz korzystać z tego atrybutu.

Listing 3.3. *Pierwsza strona JSP*

```
1.<html><body> Witaj,
2.<%= request.getAttribute("name") %>
3.</body></html>
4.<html>
5.<body>
6.<% User u = (User) request.getAttribute("użytkownik"); %>
7.Użytkownikiem jest: <%= u.getName() %>
8.</body>
9.</html>
```

Oddzielenie widoku od logiki biznesowej zależy od atrybutów, tak jak przedstawiono w listingach 3.2 i 3.3. W ten sposób technologia JSP rozwiązała problem krzyżowania się w serwletach warstwy prezentacji z logiką biznesową. Jak widać jednak w listingu 3.4, wraz z wymieszaniem skrypletów (kod Java) w kodzie prezentacji (JSP) logika biznesowa zaczęła krzyżować się z zagadnieniem prezentacji.

Listing 3.4. *Korzystanie ze skrypletów i wyrażeń w technologii JSP*

```
1.<% User u = new User(); %>
2.Użytkownikiem jest: <%= u.getName() %>
```

- **Wiersz 1.** — skryplet tworzy instancję klasy o nazwie `User`.
- **Wiersz 2.** — wyświetla nazwę użytkownika za pomocą wyrażenia (ang. *expression*).

Co poszło nie tak? Skryplety i wyrażenia przeniosły kod Java na stronę JSP. Przed technologią JSP kod prezentacji krzyżował się z kodem biznesowym. Przy zastosowaniu JSP kod biznesowy krzyżuje się z kodem prezentacji. Tak więc technologia JSP w niezamierzony sposób nie rozwiązuje niczego, ale odwraca do góry nogami problem nakładania się logiki biznesowej i logiki prezentacji. Skryplet i wyrażenie zawarte w listingu 3.4 rzeczywiście mogą być łatwo zastąpione standardowymi akcjami JSP (`<useBean>`), tak jak przedstawiono w listingu 3.5.

Listing 3.5. *Korzystanie ze standardowych akcji JSP oraz języka wyrażeń EL*

```
1.<jsp:useBean id="user" class="com.apress.User"/>
2.Użytkownikiem jest: ${user.name}
```

Listing 3.5 odgrywa tę samą rolę co listing 3.4, ale bez użycia kodu Java wewnątrz strony JSP.

- **Wiersz 1.** — tworzy instancję klasy `User` za pomocą standardowej akcji JSP.
- **Wiersz 2.** — wprowadza kluczową funkcję języka wyrażeń EL zwaną **wyrażeniem EL**, które zastępuje element skryptowy nazywany **wyrażeniami**. Składnię przedstawioną w wierszu 2. omówimy bardziej szczegółowo w poniższych punktach.

Ogólnie rzecz biorąc, standardowe akcje JSP są zbyt ograniczone. W konsekwencji programiści musieli uciekać się do korzystania ze skrypletów, aby tworzyć bogate w funkcje aplikacje sieciowe. Korzystanie z kodu Java w formie skrypletów w technologii JSP prowadzi do powstawania nieutrzymywalnych stron JSP. W rezultacie specyfikacja technologii JSP ewoluowała, aby móc obsługiwać strony JSP wolne od kodu Java. Wsparcie to opiera się przede wszystkim na języku wyrażeń EL JSP (ang. *Expression Language JSP*) oraz standardowej bibliotece znaczników (ang. *Standard Tag Library — JSTL*). W kolejnych punktach przyjrzymy się bliżej językowi wyrażeń EL oraz bibliotece JSTL.

Język wyrażeń EL

Podobnie jak piękno bez wyrazu jest martwe, tak technologia JSP bez języka wyrażeń EL jest chaosem. Podstawową zasadą języka wyrażeń EL jest zapewnienie bezskryptowych komponentów stron JSP. Język wyrażeń EL jest używany na dwa sposoby:

- Do pobierania obiektów z atrybutów objętych zakresami (zostało to wyjaśnione w poprzednim rozdziale). Są to obiekty takie jak JavaBeans, mapy, tablice i listy, które są przechowywane jako atrybuty w jednym z czterech zakresów (to również zostało wyjaśnione w poprzednim rozdziale). Język wyrażeń EL wyszukuje najpierw atrybuty w najmniejszym zakresie, który jest zakresem strony. Następnie wyszukuje je w zakresie żądania i sesji, a na koniec w największym zakresie, który jest zakresem aplikacji.
- Do uzyskiwania dostępu do parametrów żądania, nagłówków żądania, ciasteczek, parametrów inicjowania kontekstu i obiektów `pageContext`.

Wyrażenie EL jest tworzone poprzez użycie konstruktu `${wyrażenie}` lub `#{wyrażenie}`. Mimo że oba konstrukty są ewaluowane w taki sam sposób przez język wyrażeń EL, to konstrukt `${wyrażenie}` służy do przeprowadzania natychmiastowej ewaluacji, a konstrukt `#{wyrażenie}` służy do przeprowadzania ewaluacji opóźnionej:

- **Natychmiastowa ewaluacja**: wyrażenie jest kompilowane, gdy kompilowana jest strona JSP. Następnie wyrażenie jest wykonywane, gdy wykonywana jest strona JSP.
- **Opóźniona ewaluacja**: wyrażenie nie jest poddawane ewaluacji, dopóki jego wartość nie jest wymagana przez system.

■ **Uwaga** Zwróć uwagę, że w technologii JSP w wersji 2.1 i kolejnych wyrażenia `#{}` są dozwolone tylko dla atrybutów znaczników, które akceptują opóźnione wyrażenia. Wyrażenie `#{wyrażenie}` wygeneruje błąd, jeśli zostanie wykorzystane w innym dowolnym miejscu.

W kolejnych punktach przyjrzymy się składni języka wyrażeń EL oraz zastrzeżonym słowom tego języka i zobaczymy, jak używać go na stronach JSP. Gdy już poznasz podstawy, nauczysz się korzystać z języka wyrażeń EL do odczytu wartości z obiektów JavaBeans. Następnie w końcowej części rozdziału dowiesz się, jak korzystać z funkcji języka wyrażeń EL.

Literały

Literał języka wyrażeń EL może być wartością logiczną (ang. *boolean*), liczbą całkowitą (ang. *integer*), liczbą zmiennoprzecinkową (ang. *floating-point*), łańcuchem znaków (ang. *string*) lub wartością zerową (ang. *null*). W tabeli 3.1 przedstawiono poprawne wartości dla każdego typu literału.

Tabela 3.1. *Literały w języku wyrażeń EL*

Typ literału	Poprawna wartość literału
Wartość logiczna	`true` (prawda) lub `false` (fałsz)

Liczba całkowita	`-11`
	`0`
	`12345`

Liczba zmiennoprzecinkowa	`4.21`
	`-8.01`
	`1.0E12`
	`0.12`

Łańcuch znaków	Obie formy są poprawne: `"witaj!"` oraz `'witaj!'`
Wartość zerowa	`null`

Słowa zastrzeżone

Jak w przypadku każdego innego języka, język wyrażeń EL zawiera słowa, które są zastrzeżone i nie powinny być wykorzystywane jako identyfikatory. Tabela 3.2 zawiera listę słów zastrzeżonych w języku wyrażeń EL.

Tabela 3.2. *Słowa zastrzeżone w języku wyrażeń EL*

`and`	**`eq`**	**`gt`**	**`true`**
`instanceof`	**`or`**	**`ne`**	**`le`**
`false`	**`empty`**	**`not`**	**`lt`**
`ge`	**`null`**	**`div`**	**`mod`**

Operatory języka wyrażeń EL

Operatory języka wyrażeń EL są niezbędne do obsługi manipulacji danymi. Język wyrażeń EL obsługuje kilka operatorów, takich jak operatory relacyjne, arytmetyczne, logiczne itd.

Operatory arytmetyczne

W listingu 3.6 przedstawiono niektóre z tego typu operatorów. Możesz użyć tego kodu w pliku JSP i uruchomić go na serwerze. Na rysunku 3.2 pokazano dane wyjściowe.

Listing 3.6. *Operatory arytmetyczne*

```
<table border="1">
<tr>
<td><b>Operator arytmetyczny</b></td>
<td><b>Wynik typu Boolean</b></td>
</tr>
<tr>
<td>${'${'}2 + 2 }</td>
<td>${2 + 2}</td>
</tr>
<tr>
<td>${'${'}2 – 2 }</td>
<td>${2 – 2}</td>
</tr>
<tr>
<td>${'${'}2 * 2 }</td>
<td>${2 * 2}</td>
</tr>
<tr>
<td>${'${'}2 / 2 }</td>
<td>${2 / 2}</td>
</tr>
<tr>
<td>${'${'}2 mod 2 }</td>
<td>${2 mod 2}</td>
</tr>
</table>
```

Operator arytmetyczny	Wynik typu Boolean
${2 + 2 }	4
${2 - 2 }	0
${2 * 2 }	4
${2 / 2 }	1.0
${2 mod 2 }	0

Rysunek 3.2. *Operatory arytmetyczne*

Operatory relacyjne

Operatory te obejmują `==`, `!=`, `<`, `>`, `<=`, `>=`, `eq`, `ne`, `lt`, `gt`, `le` oraz `ge`.

W listingu 3.7 przedstawiono wszystkie tego typu operatory. Możesz użyć tego kodu w pliku JSP i uruchomić go na serwerze. Dane wyjściowe zostały przedstawione na rysunku 3.3.

Listing 3.7. *Operatory relacyjne*

```
<table border="1">
<tr>
<td><b>Operator relacyjny</b></td>
<td><b>Wynik typu Boolean</b></td>
</tr>
<tr>
<td>${'${'}10 &lt; 20}</td>
<td>${10 < 20}</td>
</tr>
<tr>
<td>${'${'}10 &gt; 20}</td>
<td>${10 > 20}</td>
</tr>
<tr>
<td>${'${'}10 &gt;= 10}</td>
<td>${10 >= 10}</td>
</tr>
<tr>
<td>${'${'}10 &lt;= 10}</td>
<td>${10 <= 10}</td>
</tr>
<tr>
<td>${'${'}10 == 10}</td>
<td>${10 == 10}</td>
</tr>
<tr>
<td>${'${'}10 != 20}</td>
<td>${10 != 20}</td>
</tr>
<tr>
<tr>
<td>${'${'}10 lt 20}</td>
<td>${10 lt 20}</td>
</tr>
<tr>
<td>${'${'}10 gt 20}</td>
<td>${10 gt 20}</td>
</tr>
<tr>
<td>${'${'}10 le 10}</td>
<td>${10 le 10}</td>
</tr>
<tr>
<td>${'${'}10 ge 10}</td>
<td>${10 ge 10}</td>
</tr>
<tr>
<td>${'${'}10 eq 10}</td>
<td>${10 eq 10}</td>
</tr>
<tr>
<td>${'${'}10 ne 20}</td>
<td>${10 ne 20}</td >
</tr>
</table>
```

Operator relacyjny	Wynik typu Boolean
${10 < 20}	true
${10 > 20}	false
${10 >= 10}	true
${10 <= 10}	true
${10 == 10}	true
${10 != 20}	true
${10 lt 20}	true
${10 gt 20}	false
${10 le 10}	true
${10 ge 10}	true
${10 eq 10}	true
${10 ne 20}	true

Rysunek 3.3. *Operatory relacyjne*

Operatory logiczne

W listingu 3.8 przedstawiono w akcji operatory logiczne, takie jak `&&`, `||` oraz operator `not`. Możesz użyć tego kodu w pliku JSP i uruchomić go na serwerze. Dane wyjściowe zostały przedstawione na rysunku 3.4.

Listing 3.8. *Operatory logiczne*

```
<table border="1">
<tr>
<td><b>Operator logiczny</b></td>
<td><b>Wynik</b></td>
</tr>
<tr>
<td>${'${'}true && false}</td>
<td>${true && false}</td>
</tr>
<tr>
<td>${'${'}true || false}</td>
<td>${true || false}</td>
</tr>
<tr>
<td>${'${'}not true}</td>
<td>${not true}</td>
</tr>
</table>
```

Operator logiczny	Wynik
${true && false}	false
${true || false}	true
${not true}	false

Rysunek 3.4. *Operatory logiczne*

Korzystanie z języka wyrażeń EL

W tym punkcie utworzymy prostą aplikację na podstawie modelu naszej aplikacji księgarni. Aplikacja ta nie tylko pokaże, jak korzystać z języka wyrażeń EL, ale również zademonstruje jego znaczenie. Na rysunku 3.5 przedstawiono zależność między klasą `Book` i `Author` w aplikacji, zaimplementowaną w listingach 3.9 oraz 3.10.

Rysunek 3.5. *Zależność pomiędzy klasą Book oraz klasą Author*

Listing 3.9. *Zawartość pliku Author.java*

```
1.package com.apress.chapter03.model;
2.
3.public class Author {
4.private String name;
5.
6.public String getName() {
7.return name;
8.}
9.
10.public void setName(String name) {
11.this.name = name;
12.}
13.
14.}
```

Listing 3.10. *Zawartość pliku Book.java*

```
1.package com.apress.chapter03.model;
2.
3.public class Book {
4.
5.private String bookTitle;
6.private Author author;
7.
8.public String getBookTitle() {
9.return bookTitle;
10.}
11.
12.public void setBookTitle(String bookTitle) {
13.this.bookTitle = bookTitle;
14.}
15.
16.public Author getAuthor() {
17.return author;
18.}
19.
20.public void setAuthor(Author author) {
21.this.author = author;
22.}
23.
24.}
```

Celem aplikacji jest przedstawienie, w jaki sposób uzyskać dostęp do właściwości atrybutu (właściwość `bookTitle` klasy `Book`, jak na rysunku 3.5) i właściwości atrybutu, która sama jest właściwością atrybutu (właściwość `name` klasy `Author`, jak na rysunku 3.5). Na rysunku 3.5 należy wyświetlić dane wyjściowe dla wartości właściwości typu `name` klasy `Author`. Nie dokonamy tego, stosując standardowe akcje JSP. Jak zobaczysz w dalszej części rozdziału, w takich przypadkach zostały wykorzystane skryplety. W ten sposób można zastosować skryplety w stronach JSP. Nie powinno się jednak używać skrypletów, ponieważ przy skrypletach (kod Java) przemieszanych w kodzie prezentacji (JSP) logika biznesowa krzyżuje się z zagadnieniem prezentacji, powodując powstanie nieutrzymywalnych stron JSP, jak wyjaśniono wcześniej w listingu 3.4. Ponieważ standardowe akcje JSP nie mogą uzyskać dostępu do właściwości atrybutu, która sama jest właściwością atrybutu, a stosowanie skrypletów skutkuje powstawaniem nieutrzymywalnych stron JSP, należy użyć wyrażenia EL. W tym rozdziale dowiesz się, jak używać języka wyrażeń EL na przykładzie, w którym klasa `Author` jest właściwością klasy `Book`. Na rysunek 3.5 przedstawiono zależność pomiędzy klasą `Book` oraz klasą `Author`. Będziesz mieć dostęp z poziomu klasy `Book` do właściwości obiektu `name` klasy `Author` bez wykorzystania skrypletów.

W listingu 3.9 przedstawiono obiekt `Author` z jedną właściwością o nazwie `name` z metod pobierania (ang. *getters*) i metod ustawiających (ang. *setters*). Musisz wydobyć wartość właściwości `name` klasy `Author`.

W listingu 3.10 przedstawiono obiekt `Book` z dwiema właściwościami, `bookTitle` oraz `author`, a także ich obiekty pobierające i ustawiające. Właściwością `author` w obiekcie `Book` jest klasa `Author`, tak jak przedstawiono wcześniej w listingu 3.9. Musisz uzyskać dostęp do właściwości `name` właściwości `author`.

Obiekty `Author` oraz `Book` służą jako model aplikacji MVC (ang. *Model-View-Controller*), czyli model-widok-kontroler. W listingu 3.11 przedstawiono kontroler aplikacji.

Listing 3.11. *Zawartość pliku BookController.java*

```
1.package com.apress.chapter03.controller;
2.
3.import java.io.IOException;
4.
5.import javax.servlet.RequestDispatcher;
6.import javax.servlet.ServletException;
7.import javax.servlet.http.HttpServlet;
8.import javax.servlet.http.HttpServletRequest;
9.import javax.servlet.http.HttpServletResponse;
10.
11.import com.apress.chapter03.model.Author;
12.import com.apress.chapter03.model.Book;
13.
14.public class BookController extends HttpServlet {
15.
16.protected void doGet(HttpServletRequest request,
17.HttpServletResponse response) throws ServletException, IOException {
18.Book book = new Book();
19.book.setBookTitle("Learning Java Web");
20.Author author = new Author();
21.author.setName("Vishal Layka");
22.book.setAuthor(author);
23.
24.request.setAttribute("bookAttrib", book);
25.
26.RequestDispatcher view = request.getRequestDispatcher("/book.jsp");
27.view.forward(request, response);
28.}
29.
30.}
```

Listing 3.11 jest częścią kontrolera dla wzorca MVC. Jak dowiedzieliśmy się w listingu 3.2, separacja zagadnienia widoku od zagadnienia logiki biznesowej zależy od atrybutów. W związku z tym trzeba zapisać obiekt modelu do atrybutów dla widoku (JSP), aby móc uzyskać dostęp do modelu za pomocą atrybutów.

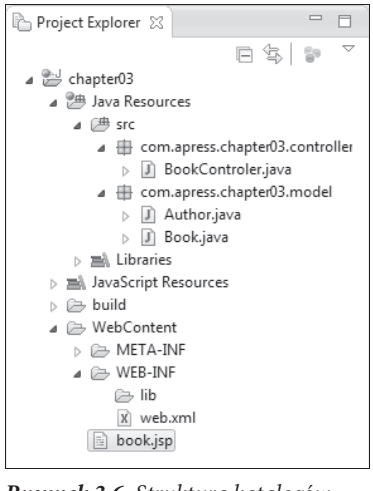
- **Wiersze 19. – 22.** — w tych wierszach można ustawić właściwości `bookTitle` i `author` należące do obiektu `Book`. Należy pamiętać, że właściwość `name` klasy `Author` jest już ustawiona w wierszu 21.
- **Wiersz 22.** — ustawia właściwość `author` należącą do `book`.
- **Wiersz 24.** — ustawia także obiekt `Book` jako atrybut w żądaniu.
- **Wiersze 26. – 27.** — wiersz 26. powinien wyglądać znajomo. W tym wierszu wysyłasz żądanie do pliku *book.jsp*.

Listing 3.12 dostarcza deskryptor wdrażania dla tej aplikacji.

Listing 3.12. *Zawartość pliku web.xml*

```
1.<?xml version="1.0" encoding="UTF-8"?>
2.<web-app xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance "
3. xmlns=" http://java.sun.com/xml/ns/javaee "
4. xmlns:web=" http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd "
5. xsi:schemaLocation=" http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd " id="WebApp_ID" version="3.0">
7.<display-name>chapter03</display-name>
8.<servlet>
9.<servlet-name>BookController</servlet-name>
10.<servlet-class>com.apress.chapter03.controller.BookController</servlet-class>
11.</servlet>
12.<servlet-mapping>
13.<servlet-name>BookController</servlet-name>
14.<url-pattern>/book</url-pattern>
15.</servlet-mapping>
16.<welcome-file-list>
17.<welcome-file>index.html</welcome-file>
18.</welcome-file-list>
19.</web-app>
```

Teraz jedynym brakującym elementem kluczowym w tej aplikacji sieciowej jest strona JSP, w której zapisany jest kod języka wyrażeń EL. Zanim przejdziemy do strony JSP, przyjrzymy się dwóm kluczowym operatorom dostarczanym przez język wyrażeń EL w celu uzyskania dostępu do zhermetyzowanych danych: `[]` oraz `.`, nazywany również *dot* (kropka). Po zdobyciu wiedzy na temat działania operatorów `[]` oraz `.` będziesz w stanie napisać kod pliku *book.jsp*. Na rysunku 3.6 przedstawiono strukturę katalogów aplikacji.



Rysunek 3.6. *Struktura katalogów*

Operatory [ ] oraz .

Korzystanie z zapisu . jest skrótem dostępu do właściwości obiektu. Operator dot został wprowadzony w listingu 3.5. Dla ułatwienia zostało to ponownie przedstawione w listingu 3.13.

Listing 3.13. *Korzystanie z operatora dot*

```
1.<jsp:useBean id="user" class="com.apress.User"/>
2.Użytkownikiem jest: ${user.name}
```

- **Wiersz 1.** — klasa `User` jest tworzona przy użyciu akcji `<jsp:useBean>`. Obiekt `user` został ustawiony jako atrybut `request` w kodzie serwletu.
- **Wiersz 2.** — za pomocą wyrażenia `${user.name}` uzyskiwany jest dostęp do wartości nazwy, która jest właściwością obiektu `User`.

Zmienna `user` w wyrażeniu `${user.name}` jest atrybutem przechowywanym w zakresie `request`. Zmienna w wyrażeniu EL, do którego stosowany jest operator dot, może być jednym z dwóch typów:

- atrybutem, który jest przechowywany w jednym z czterech zakresów, tak jak w tym przypadku;
- niejawnym obiektem języka wyrażeń EL, co zostało wyjaśnione w dalszej części tego rozdziału.

Niezależnie od tego, czy zmienna jest niejawnym obiektem języka wyrażeń EL, czy atrybutem przechowywanym w jednym z czterech zakresów, może być ona obiektem JavaBean lub mapą. W listingu 3.13 zmienna ta jest obiektem JavaBeans, który został ustawiony jako atrybut w zakresie `request`, więc `name` jest właściwością obiektu JavaBeans, do którego odnosi się zmienna `user`. Gdyby zmienna `user` była mapą ustawioną jako atrybut w jednym z czterech zakresów, nazwa byłaby kluczem mapy.

Uwaga Zmienna w wyrażeniu EL, dla której jest stosowany operator dot, mogłaby być atrybutem ustawionym w dowolnym z czterech zakresów lub niejawnym obiektem języka wyrażeń EL. Co więcej, niezależnie od tego, czy ta zmienna jest atrybutem ustawionym w jednym z czterech zakresów, czy niejawnym obiektem języka wyrażeń EL, może ona być obiektem JavaBeans lub mapą. Jeśli zmienna jest obiektem JavaBeans, to po niej następuje jej właściwość zaraz po operatorze dot. Jeśli zmienna jest mapą, to po niej następuje jej klucz zaraz po operatorze dot.

Teraz powinno być jasne, że zmienna w wyrażeniu języka wyrażeń EL jest obiektem JavaBeans lub mapą. Ale co zrobić, jeśli chcesz ustawić tablicę jako atrybut w jednym z czterech zakresów i uzyskać dostęp do jej elementów za pomocą wyrażenia EL? Albo co zrobić, jeśli chcesz ustawić listę jako atrybut w jednym z czterech zakresów i uzyskać dostęp do jej elementów za pomocą wyrażenia EL? Odpowiedzią jest operator [] dostarczany przez język wyrażeń EL.

Operator [] jest używany do uzyskiwania dostępu do tablic, list, obiektów JavaBeans i map. Oznacza to, że zmienne, do których stosuje się operator [], mogą być tablicą, listą, obiektem JavaBeans lub mapą.

Zawartość nawiasów

W nawiasie operatora [] może znajdować się jedna z następujących zawartości:

- Indeks z cudzysłowem lub bez cudzysłowu.
- Literał w postaci łańcucha znaków.
- Niejawny obiekt języka wyrażeń EL lub atrybut w jednym z czterech zakresów. Niejawne obiekty języka wyrażeń EL zostały omówione w dalszej części tego rozdziału.
- Zagnieżdżone wyrażenie.

Jeśli istnieje indeks z cudzysłowem lub bez cudzysłowu wewnątrz nawiasów operatora [], zmienna, do której stosuje się operator [], jest tablicą lub listą. W listingu 3.14 przedstawiono, jak korzystać z operatora [] z listą lub tablicą.

Listing 3.14. *Użycie operatora [ ] z obiektem listy lub tablicy*

```
someArray["1"]
someArray[1]
someList["2"]
someList[2]
```

Do tablicy z listingu 3.15 można uzyskać dostęp np. w taki sposób, jak przedstawiono w listingu 3.16.

Listing 3.15. *Ustawianie tablicy jako atrybutu w kodzie serwletu*

```
1. String [ ] books = {"Clojure", "Groovy ", "Java" , "Scala"} ;
2. request.setAttribute("books", books);
```

Listing 3.16. *Korzystanie z operatora []*

```
Książka: $ {books [0]}
```

Wynik działania kodu z listingu 3.16 jest wyświetlany w następujący sposób:

```
Książka: Clojure
```

■ **Uwaga** Dostęp do list można uzyskać w ten sam sposób jak do tablic.

Jeśli w nawiasach operatora [] znajduje się literał w postaci łańcucha znaków, zmienna, do której operator [] jest stosowany, jest obiektem JavaBeans lub mapą. W listingu 3.17 przedstawiono kod do ustawienia mapy jako atrybutu w serwlecie.

Listing 3.17. *Fragment kodu do ustawienia mapy jako atrybutu w kodzie serwletu*

```
1.Map<String, String> bookMap = new HashMap<>();
2.bookMap.put("Groovy", "Beginning Groovy");
3.bookMap.put("Java", " Beginning Java");
4.bookMap.put("Scala", " Beginning Scala");
5.request.setAttribute("books", bookMap);
```

W listingu 3.18 język wyrażeń EL wyszukuje atrybut związany z nazwą obiektu `books` w zakresie. W listingu 3.17 obiekt `books` jest mapą ustawioną w atrybucie `request`. Dlatego język wyrażeń EL wyszukuje słowo kluczowe `Groovy` wprowadzone w operatorze [] w listingu 3.18 i przeprowadza jego ewaluację.

Listing 3.18. *Korzystanie z operatora []*

```
Książka: $ {books["Groovy"]}
```

Wynik działania kodu z listingu 3.18 wyświetlany jest w następujący sposób:

```
Książka: Beginning Groovy
```

Operatory `.` oraz [] mogą być używane z obiektem JavaBeans lub obiektem mapy. W listingu 3.18 można np. wykorzystać operator `.`, tak jak przedstawiono poniżej:

```
$ {books.Groovy}
```

Jeśli wewnątrz nawiasów operatora [] nie znajdują się literał w postaci łańcucha znaków ani indeks z cudzysłowem lub bez cudzysłowu oraz jeśli zawartość nawiasów operatora [] nie jest niejawnym obiektem języka wyrażeń EL, to dana zawartość jest poddawana ewaluacji przez wyszukiwanie atrybutu o tej nazwie w jednym z czterech zakresów. Zostało to przedstawione w listingach 3.19 oraz 3.20.

Listing 3.19. *Fragment kodu do ustawienia mapy jako atrybutu kodu serwletu*

```
1.Map<String, String> bookMap = new HashMap<>();
2.bookMap.put("Groovy", "Beginning Groovy");
3.bookMap.put("Java", " Beginning Java");
4.bookMap.put("Scala", " Beginning Scala");
5.request.setAttribute("books", bookMap);
6.request.setAttribute("java", "Java");
```

Listing 3.20. *Korzystanie z operatora []*

```
Książka : ${books [java]}
```

Przyjrzyjmy się, jak działa ewaluacja w listingu 3.20.

- W kodzie wyrażenia `${books[java]}` w listingu 3.20 język wyrażeń EL wyszukuje atrybut związany z nazwą `books` w zakresach.
- Język wyrażeń EL znajduje ten atrybut w zakresie żądania, ponieważ parametr `books` został ustawiony jako atrybut żądania w wierszu 5. w listingu 3.19.
- W listingu 3.20 zawartością operatora [] jest obiekt `java`, który nie jest ani literałem w postaci łańcucha znaków, ani niejawnym obiektem języka wyrażeń EL. W związku z tym język wyrażeń EL wyszukuje w zakresach atrybut związany z nazwą `java` i znajduje go w zakresie żądania, ponieważ obiekt `java` był ustawiony jako atrybut żądania w wierszu 6. w listingu 3.19.
- Korzystając z wartości `Java` obiektu `java` w wierszu 6. w listingu 3.19, wyrażenie EL staje się wyrażeniem `${books["Java"]}`.
- Ponieważ atrybut `books` jest mapą ustawioną jako atrybut żądania w wierszu 5. w listingu 3.19, język wyrażeń EL szuka klucza `Java`, który znajduje się w wierszu 3. w listingu 3.19 i wyświetla jego wartość, tak jak przedstawiono poniżej:

```
Książka: Beginning Java
```

Jeżeli w nawiasach operatora [] znajduje się wyrażenie EL, zawartość tych nawiasów (w tym przypadku wyrażenie EL) jest poddawana ewaluacji według tych samych zasad, które są stosowane do każdego wyrażenia EL. Innymi słowy jeśli wyrażenie EL korzysta z operatora *dot* lub operatora [], stosowane są te same zasady, które zostały omówione wcześniej. Zostało to przedstawione w listingach 3.21 oraz 3.22.

Listing 3.21. *Fragment kodu dla ustawienia mapy i tablicy jako atrybutów w kodzie serwletu*

```
1.Map<String, String> bookMap = new HashMap<>();
2.bookMap.put("Groovy", "Beginning Groovy");
3.bookMap.put("Java", " Beginning Java");
4.bookMap.put("Scala", " Beginning Scala");
5.request.setAttribute("books", bookMap);
6.
7.String[ ] categories = {"Groovy", "Java", "Scala"};
8.request.setAttribute("category", categories);
```

Listing 3.22. *Zagnieżdżone wyrażenie EL*

```
Książka : ${ books[category[1]] }
```

Oto dane wyjściowe:

```
Książka: Beginning Java
```

Ponieważ nauczyłeś się, w jaki sposób korzystać z operatorów `.` oraz `[]`, nadszedł czas, aby wyjaśnić, dlaczego język wyrażeń EL jest tak ważny. Dowiesz się tego, kończąc aplikację, którą zacząłeś tworzyć, a w tym celu napiszesz stronę JSP. W listingu 3.23 przedstawiono zawartość pliku *book.jsp*. Ta strona wykorzystuje elementy skryptowe JSP (skryplety i wyrażenia) oraz język wyrażeń EL. Chodzi o to, aby porównać np. elementy skryptowe i język wyrażeń EL. Porównanie to zostało przedstawione na rysunku 3.7.

Listing 3.23. *Zawartość pliku book.jsp*

```
1.<%@page import="com.apress.chapter03.model.Book"%>
2.<%@page import="com.apress.chapter03.model.Author"%>
3.<%@ taglib uri=" http://java.sun.com/jsp/jstl/functions " prefix="fn"%>
4.<%@ taglib uri=" http://java.sun.com/jstl/core " prefix="c"%>
5.<html>
6.<head>
7.</head>
8.<body>
9.<table border="1">
10.<tr>
11.<th width= "20px">Opis</th>
12.<th >kod</th>
13.<th >dane wyjściowe</th>
14.</tr>
15.<%
16.Book book = (Book) request.getAttribute("bookAttrib");
17.Author author = book.getAuthor();
18.%>
19.<tr>
20.<td>Imię autora przy użyciu <b>skrypletu i wyrażenia</b>
21.</td>
22.<td>${fn:escapeXml("<%= author.getName() %>")}</td>
23.<td><%=author.getName()%></td>
24.</tr>
25.
26.<jsp:useBean id="bookAttrib" class="com.apress.chapter03.model.Book"
27.scope="request" />
28.<tr>
29.<td>Imię autora przy użyciu <b> akcji jsp:getProperty </b>
30.</td>
31.<td>
32.<table border="1">
33.<tr>
34.<td>${fn:escapeXml("<jsp:getProperty name = \"bookAttrib \" property= \"author \" />")}</td>
35.</tr>
36.<tr>
37.<td>${fn:escapeXml("<jsp:getProperty name = \"bookAttrib \" property= \"author.name \" />")}</td>
38.</tr>
39.</table>
40.
41.</td>
42.<td>
43.<table border="1">
44.<tr>
45.<td><jsp:getProperty name="bookAttrib" property="author" />
46.</td>
47.</tr>
48.<tr>
49.<td>
50.<%-- <jsp:getProperty name ="bookId" property="author.name" /> — ten kod wygeneruje wyjątek podczas uruchomienia--%>
```

```
51.Niemożliwe
52.</td>
53.</tr>
54.</table>
55.</td>
56.</tr>
57.<tr>
58.<td>Imię autora przy użyciu<b> wyrażenia EL </b></td>
59.<td>${fn:escapeXml("${bookAttrib.author.name}")}</td>
60.<td>${bookAttrib.author.name}</td>
61.</tr>
62.</table>
63.</body>
64.</html>
```

- **Wiersz 23.** — wyświetla nazwisko autora za pomocą wyrażeń.
- **Wiersz 50.** — pokazuje, że nie jest możliwe wyświetlenie nazwiska autora za pomocą standardowej akcji JSP.
- **Wiersz 60.** — wyświetla nazwisko autora, używając języka wyrażeń EL.

Na rysunku 3.7 przedstawiono to, co można zobaczyć po uruchomieniu tej aplikacji (*http://localhost:8080/chapter03/book*). W zasadzie nie jest możliwe, aby wyświetlić wartość właściwości parametru `name` w obiekcie `Author` przy użyciu standardowych akcji JSP. Elementy skryptowe byłyby jedynym sposobem, aby to zrobić, gdyby tylko nie chodziło o język wyrażeń EL.

Opis	Kod	Dane wyjściowe
Imię autora przy użyciu **skrypletu i wyrażenia**	<%= author.getName() %>	Vishal Layka
Imię autora przy użyciu **akcji jsp:getProperty**	<jsp:getProperty name = "bookAttrib" property="author" />	com.apress.r03.model.Author@c43be4
	<jsp:getProperty name = "bookAttrib" property="author.name" />	Niemożliwe
Imię autora przy użyciu **wyrażenia EL**	${bookAttrib.author.name}	Vishal Layka

Rysunek 3.7. *Porównanie skrypletów, standardowych akcji oraz języka wyrażeń EL*

Obiekty niejawne języka wyrażeń EL

Skryplety mają dostęp do kilku obiektów niejawnych JSP, co zostało wyjaśnione w rozdziale 2. Obiekty te umożliwiają uzyskanie dostępu do wszelkich zmiennych, które są przechowywane w poszczególnych zakresach JSP. Język wyrażeń EL dostarcza również własne obiekty niejawne zwane **obiektami niejawnymi EL** (ang. *EL implicit objects*). Obiekty niejawne EL nie są takie same jak obiekty niejawne JSP (z wyjątkiem `pageContext`). Wszystkie te niejawne obiekty EL są mapami, które mapują odpowiedni zakres nazw atrybutów na ich wartości. Stosując np. obiekty niejawne typu `param` i `paramValues`, można uzyskać dostęp do parametrów żądania HTTP. W tabeli 3.3 opisano obiekty niejawne EL.

Tabela 3.3. *Obiekty niejawne EL*

Obiekt niejawny	Opis
`cookie`	Mapa: mapuje nazwy plików cookie na pojedynczy obiekt `Cookie`.
`header`	Mapa: zawiera wartości każdej nazwy nagłówka.
`headerValues`	Mapa: mapuje nazwę nagłówka ma tablicę łańcuchów znaków wszystkich możliwych wartości nagłówka.
`initParam`	Mapa: mapuje nazwy parametrów inicjacji kontekstu na ich wartości parametrów łańcucha znaków.
`param`	Mapa: zawiera nazwy parametrów dla strony.
`paramValues`	Mapa: mapuje nazwę parametru na tablicę łańcuchów znaków wszystkich wartości parametru.
`pageContext`	Obiekt typu `PageContext`.
`applicationScope`	Mapa: zawiera wszystkie zmienne z zakresu aplikacji.
`pageScope`	Mapa: zawiera wszystkie zmienne z zakresu strony.
`requestScope`	Mapa: zawiera wszystkie zmienne z zakresu żądania.
`sessionScope`	Mapa: zawiera wszystkie zmienne z zakresu sesji.

Korzystanie z niejawnych obiektów EL

Obiekty niejawne EL wymienione w tabeli 3.3, takie jak `applicationScope`, `pageScope`, `requestScope` i `sessionScope`, służą do określania zakresu. Są one wykorzystywane do uzyskiwania dostępu do atrybutów objętych zakresem (ang. *scoped attributes*), czyli uzyskiwania dostępu do danych z obiektów JavaBeans, map, tablic i list, które były przechowywane jako atrybuty w jednym z czterech zakresów: strony, żądania, sesji oraz aplikacji.

Pozostałe obiekty niejawne wymienione w tabeli 3.3 są wykorzystywane do uzyskiwania dostępu do parametrów żądania, nagłówków żądania, ciasteczek, parametrów inicjacji kontekstu oraz obiektów `pageContext`. W tym punkcie przedstawiono zastosowanie niektórych obiektów niejawnych EL.

Uzyskiwanie dostępu do parametrów żądania

W listingu 3.24 przedstawiono prosty formularz stosowany do wysyłania żądania parametrów za pomocą pliku *form.jsp*.

Listing 3.24. *Zawartość pliku form.jsp*

```
1.<body>
2.<form action="books" method="post">
3.<input type="hidden" name="action" value="books"/>
4.<p>Tytuł książki: <input type="text" name="bookTitle"></p>
5.<p>Imię autora 1.: <input type="text" name="authorName"></p>
6.<p>Imię autora 2.: <input type="text" name="authorName"></p>
7.
8.<input type = "submit"/>
9.</form>
10.</body>
```

W listingu 3.24 właściwość `name` znacznika `<input>` jest taka sama, czyli `authorName`.

W listingu 3.25 przedstawiono sposób użycia obiektów niejawnych EL typu `param` i `paramValues` w celu pobrania parametru żądania i wyświetlenia rezultatu na stronie *result.jsp*.

Listing 3.25. *Zawartość pliku result.jsp*

```
1.<p>Tytuł książki: ${param.bookTitle}<br>
2.Autor 1.: ${paramValues.authorName[0]}<br>
3.Autor 2.: ${paramValues. authorName[1]}
4.</p>
```

- **Wiersz 1.** — użycie niejawnego obiektu EL typu `param` w celu uzyskania tytułu obiektu `Book`.
- **Wiersze 2. – 3.** — ten kod wykorzystuje ukryte obiekty EL typu `paramValues`, aby uzyskać nazwiska autorów `Autor 1` i `Autor 2`.

Uzyskiwanie dostępu do nagłówka

Obiekty niejawne EL typu `header` i `headerValues` dają dostęp do wartości nagłówków, które można uzyskać za pomocą metod `request.getHeader()` oraz `request.getHeaders()`.

W listingu 3.26 przedstawiono uzyskiwanie dostępu do nagłówka o nazwie `user-agent` za pomocą wyrażenia `$ {header.user-agent}` lub `${header ["user-agent"]}`.

Listing 3.26. *Korzystanie z nagłówka obiektu niejawnego EL*

```
<span>${header["user-agent"]}</span>
```

Oto dane wyjściowe:

```
Mozilla/5.0 (Windows NT 6.1; rv:12.0) Gecko/20100101 Firefox/12.0
```

Uzyskiwanie dostępu do ciasteczek

Niejawny obiekt EL typu ciasteczko daje nam dostęp do pliku ciasteczka (ang. *cookie*). W listingu 3.27 przedstawiono ciasteczko przechowywane w serwlecie.

Listing 3.27. *Konfigurowanie ciasteczka w serwlecie*

```
1.String userName = "Vishal";
2.Cookie c = new Cookie("userName", userName);
3.c.setPath("/");
4.response.addCookie(c);
```

W listingu 3.28 przedstawiono, jak korzystać z niejawnego obiektu EL, aby uzyskać dostęp do pliku ciasteczka na stronie JSP.

Listing 3.28. *Korzystanie z ciasteczka obiektu niejawnego EL*

```
${cookie.userName.value}
```

Uzyskiwanie dostępu do atrybutu objętego zakresem

Obiekt niejawny EL typu `sessionScope` umożliwia uzyskanie dostępu do atrybutów przechowywanych w zakresie sesji. W listingu 3.29 przedstawiono atrybut przechowywany w sesji w serwlecie.

Listing 3.29. *Konfigurowanie atrybutu sesji w serwlecie*

```
HttpSession session = request.getSession();
Book book = new Book();
book.setBookTitle("Beginning Java");
session.setAttribute("book", book);
```

W listingu 3.30 przedstawiono korzystanie z niejawnego obiektu EL o nazwie `sessionScope` w celu uzyskania dostępu do tytułu książki na stronie JSP.

Listing 3.30. *Korzystanie z niejawnego obiektu EL typu sessionScope*

```
<span>Tytuł książki w zasięgu sesji ${sessionScope.book.bookTitle}</span>
```

Funkcje języka wyrażeń EL

Funkcje języka wyrażeń EL pozwalają wywołać metodę Java ze strony JSP bez użycia skryptów. Funkcja języka wyrażeń EL jest mapowana na statyczną metodę klasy Java. To mapowanie jest określone w deskryptorze bibliotek znaczników (ang. *tag library descriptor* — TLD), który został omówiony w dalszej części tego rozdziału. W listingu 3.31 przedstawiono prostą metodę Javy, która zwraca bieżące datę i czas.

Listing 3.31. *Klasa Java z metodami publicznymi oraz statycznymi*

```
1.package com.apress.elfunction;
2.
3.import java.text.SimpleDateFormat;
4.import java.util.Calendar;
5.
6.public class Now {
7.
8.public static String now() {
9.Calendar currentDate = Calendar.getInstance();
10.SimpleDateFormat formatter = new SimpleDateFormat(
11."yyyy/MM/dd HH:mm:ss");
12.String now = formatter.format(currentDate.getTime());
13.
14.return now;
15.}
16.}
17.
```

Kluczowym wymaganiem, aby metoda Java mogła być zastosowana w funkcji języka wyrażeń EL, jest to, że metoda ta musi być publiczna (`public`) i statyczna (`static`). Trzej główni gracze w funkcji języka wyrażeń EL są następujący:

- Metoda Java zdefiniowana w klasie.
- Strona JSP, która wywołuje metodę Java za pomocą języka wyrażeń EL.
- Plik deskryptora bibliotek znaczników, który mapuje metodę w klasie Java na kod JSP wywołujący tę metodę Java.

W listingu 3.32 przedstawiono plik deskryptora bibliotek znaczników. Plik TLD jest plikiem XML, który deklaruje bibliotekę znacznika. Ten plik TLD zawiera deklarację i mapowanie jednej lub kilku funkcji języka wyrażeń EL. Każda funkcja jest podana z nazwą i konkretną metodą w klasie Java, która zaimplementuje daną funkcję.

Listing 3.32. *Deskryptor bibliotek znaczników*

```
1.<?xml version="1.0" encoding="UTF-8"?>
2.<taglib version="2.1" xmlns=" http://java.sun.com/xml/ns/javaee "
3.xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance "
4.xsi:schemaLocation=" http://java.sun.com/xml/ns/javaee
5.http://java.sun.com/xml/ns/javaee/webjsptaglibrary_2_1.xsd ">
6.<tlib-version>1.2</tlib-version>
7.<uri>elFunction</uri>
8.<function>
```

```
9.<name>now</name>
10.<function-class>
11.com.apress.elfunction.Now
12.</function-class>
13.<function-signature>
14.String now()
15.</function-signature>
16.
17.</function>
18.</taglib>
```

- **Wiersz 7.** — jest adresem URI funkcji, która będzie zastosowana w dyrektywie `taglib` na stronie JSP.
- **Wiersze 8. – 17.** — określają metodę Java, która zostanie wywołana z podpisem, oraz klasę Java, w której ta metoda jest zdefiniowana.

Metoda Java może być wywołana ze strony JSP za pomocą języka wyrażeń EL w następujący sposób: `${`*`przedrostek`*`:`*`nazwa_funkcji`*`()}`. Przedrostek przestrzeni nazw jest deklarowany za pomocą dyrektywy `taglib` na stronie JSP. W listingu 3.33 przedstawiono kod JSP.

Listing 3.33. *Wywoływanie funkcji języka wyrażeń EL na stronie JSP*

```
1.<%@ taglib prefix="elf" uri="elFunction"%>
2.<html>
3.
4.<body>${elf:now() }
5.</body>
6.</html>
```

- **Wiersz 1.** — jest dyrektywą `taglib` z przedrostkiem `elf` i adresem URI, który jest zdefiniowany w deskryptorze bibliotek znaczników.
- **Wiersz 4.** — używa przedrostka `elf` i wywołuje funkcję. Nazwa tej funkcji jest zdefiniowana w opisie bibliotek znaczników.

Biblioteka JSTL

Podstawowym celem standardowych bibliotek znaczników JSP (ang. *JSP Standard Tag Library* — JSTL) jest uproszczenie procesu tworzenia stron Java Server Pages. Jak wspomniano wcześniej, skryplety prowadzą do powstawania nieutrzymywalnych stron JSP i mogą być zastąpione przez standardowe akcje JSP. Standardowe akcje są jednak zbyt ograniczone i lepszym sposobem dla programistów Java jest tworzenie własnych niestandardowych akcji. Mimo to tworzenie akcji niestandardowych jest trudnym zadaniem. Biblioteka JSTL zapewnia takie akcje niestandardowe, które mogą obsługiwać typowe powtarzające się zadania. JSTL obejmuje szeroki zakres akcji podzielonych na poszczególne obszary funkcjonalne. W tabeli 3.4 zawarto listę obszarów funkcjonalnych wraz z adresami URI odwołującymi się do bibliotek i przedrostkami stosowanymi w specyfikacji JSTL.

Tabela 3.4. *Biblioteki znaczników JSTL*

Obszar funkcjonalny	Adres URI	Przedrostek
Znaczniki podstawowe	`http://java.sun.com/jsp/jstl/core`	`c`
Przetwarzanie XML	`http://java.sun.com/jsp/jstl/xml`	`x`
Formatowanie obsługujące standard I18N (internacjonalizacja)	`http://java.sun.com/jsp/jstl/fmt`	`fmt`
Dostęp do relacyjnej bazy danych	`http://java.sun.com/jsp/jstl/sql`	`sql`
Funkcje	`http://java.sun.com/jsp/jstl/functions`	`fn`

Wiele akcji JSTL eksportuje zmienne objęte zakresem, do których można łatwo uzyskać dostęp za pomocą języka wyrażeń EL. Jak dowiedziałeś się w poprzednim rozdziale, zmienne objęte zakresem są obiektami przechowywanymi w jednym z zakresów JSP: aplikacji, stronie, żądaniu lub sesji. Gdy akcja JSTL tworzy zmienną objętą zakresem dostępną dla jednej strony JSP lub kilku stron JSP, ma ona atrybut o nazwie `var`, który pozwala określić nazwę tej zmiennej objętej zakresem. W listingu 3.34 przedstawiono akcję `<c:set>`, dostępną w bibliotece podstawowych znaczników JSTL w celu ustawienia wartości zmiennej objętej zakresem.

Listing 3.34. *Korzystanie z akcji <c:set>*

```
<c:set var="name" value="witaj!" scope="session" />
```

Kod z listingu 3.34 ustawia zmienną o nazwie `name` za pomocą wartości `witaj!` i pozwala, aby zmienna była widoczna w zakresie sesji. Jeśli wartość zakresu nie jest określona, domyślnym zakresem jest zakres strony. W kolejnym punkcie poznasz wszystkie akcje w różnych obszarach funkcjonalnych określonych w bibliotece podstawowych znaczników.

Biblioteki podstawowych znaczników

W tabeli 3.5 opisano podstawowe akcje z biblioteki podstawowych znaczników.

Tabela 3.5. *Akcje podstawowe w bibliotece podstawowych znaczników*

Akcja	Opis
`<c:catch>`	Przechwytuje wyjątki generowane w sekcji `body` danej akcji.
`<c:choose>`	Wybiera jeden z wielu fragmentów kodu.
`<c:forEach>`	Przeprowadza iterację przez kolekcję obiektów lub iteruje stałą liczbę razy.
`<c:forTokens>`	Przeprowadza iterację przez tokeny w łańcuchu znaków.
`<c:if>`	Warunkowo wykonuje pewną funkcjonalność.
`<c:import>`	Importuje adres URL.
`<c:otherwise>`	Określa domyślną funkcjonalność w akcji `<c:choose>`.
`<c:out>`	Wysyła dane wyjściowe do bieżącego obiektu `JspWriter`.
`<c:param>`	Określa parametr adresu URL dla akcji `<c:import>` lub `<c:url>`.
`<c:redirect>`	Przekierowuje odpowiedź do określonego adresu URL.
`<c:remove>`	Usuwa zmienną objętą zakresem.
`<c:set>`	Tworzy zmienną objętą zakresem.
`<c:url>`	Tworzy adres URL z odpowiednim przepisywaniem adresu URL.
`<c:when>`	Określa jeden z kilku warunków akcji w `<c:choose>`.

Biblioteka podstawowa JSTL może być podzielona na cztery odrębne obszary funkcjonalne, tak jak zostało to opisane w tabeli 3.6.

Tabela 3.6. *Podstawowa biblioteka JSTL*

Obszary funkcjonalne	Opis
Akcje ogólnego przeznaczenia	Używane do manipulowania zmiennymi objętymi zakresem.
Akcje warunkowe	Służące do przetwarzania warunkowego w obrębie strony JSP.
Akcje iteratora	Używane do przeprowadzania iteracji przez kolekcję obiektów.
Akcje związane z adresem URL	Używane do obsługi zasobów adresów URL na stronie JSP.

Akcje ogólnego przeznaczenia

Akcje ogólnego przeznaczenia zapewniają środki do pracy ze zmiennymi objętymi zakresem. W tabeli 3.7 opisano akcje ogólnego przeznaczenia w bibliotece podstawowych znaczników.

Tabela 3.7. *Akcje ogólnego przeznaczenia*

Akcja	Opis
`<c:out>`	Wyznacza wartość wyrażenia i wysyła wynik do obiektu `JspWriter`.
`<c:set>`	Ustawia wartość zmiennej objętej zakresem lub właściwość obiektu docelowego.
`<c:remove>`	Usuwa zmienną objętą zakresem.
`<c:catch>`	Przechwytuje klasę `java.lang.Throwable` wygenerowaną przez dowolną własną akcję zagnieżdżoną.

Akcja <c:out>

Akcja `<c:out>` przeprowadza ewaluację wyrażenia i wyświetla wynik. Jest to odpowiednik składni JSP *<%=wyrażenie%>*.

Oto składnia bez sekcji `body`:

```
<c:out value="wartość" [escapeXml="{true|false}"][default="domyślna_wartość"] />
```

Oto składnia z sekcją `body`:

```
<c:out value="wartość" [escapeXml="{true|false}"]>
domyślna_wartość
</c:out>
// Pozycje w nawiasach są opcjonalne.
```

W tabeli 3.8 opisano atrybuty akcji `<c:out>`. Ponieważ w tym rozdziale biblioteka JSLT została omówiona kompleksowo, powinieneś zapoznać się ze specyfikacją JSTL[3], aby lepiej zrozumieć, w jaki sposób korzystać z atrybutów JSTL.

Tabela 3.8. *Atrybuty akcji <c:out>*

Nazwa	Typ	Opis
`value`	`Object`	To jest wyrażenie poddawane ewaluacji.
`escapeXml`	`boolean`	Określa, czy znaki `<`, `>`, `&`, `'`, `"` w otrzymanym łańcuchu znaków powinny być konwertowane do odpowiadających im kodów encji znaków. Wartość domyślna jest ustawiona jako `true`.
`default`	`Object`	To jest wartość domyślna, jeśli otrzymana wartość jest `null`.

Wartość zapisywana do obiektu `JspWriter` jest określona jako atrybut `value`. W atrybucie `value` możesz używać wyrażeń.

- `value`: wyrażenie poddawane ewaluacji jest dostarczane przez atrybut `value`, a wynik jest przekształcany na łańcuch znaków, zanim zostanie zwrócony jako część odpowiedzi.
- `default`: opcjonalnie można określić wartość domyślną, którą akcja `<c:out>` wysyła do bieżącego obiektu `JspWriter`, jeśli określona wartość jest `null` lub nie jest poprawnym wyrażeniem. Można określić wartość domyślną z atrybutem domyślnym lub w sekcji `body` akcji `<c:out>`.
- `escapeXml`: atrybut `escapeXml` określa, czy konwertować określone znaki na kody encji znaków HTML wymienionych w tabeli 3.9. Domyślnie atrybut `escapeXml` jest ustawiony na `true`. Jeśli podasz wartość `false` dla atrybutu `escapeXml`, akcja `<c:out>` nie skonwertuje tych znaków.

[3] *http://java.coe.psu.ac.th/J2EE/JSTL1.2/jstl-1_2-mrel2-spec.pdf*

Tabela 3.9. *Kody encji znaków*

Znak	Kod encji znaku
<	<
>	>
&	&
'	&#039
"	&#034

W listingu 3.35 przedstawiono zastępowanie elementów skryptowych akcją <c:out>.

Listing 3.35. *Porównanie akcji <c:out> i wyrażenia*

```
<%= "witaj!" %>              // Dana wyjściowa "witaj!" przy użyciu wyrażenia
<c:out value = "witaj!"/> // Dana wyjściowa "witaj!" przy użyciu akcji <c:out>
```

Akcja <c:set>

Biblioteki JSTL ustawiają znacznik lub akcja <c:set> ustawia wartość zmiennej objętej zakresem albo właściwość obiektu docelowego. Akcja <c:set> jest lepszą alternatywą dla korzystania z akcji JSP <jsp:setProperty>. Odmiennie niż akcja <jsp:setProperty>, która pozwala ustawić tylko właściwość ziarna (ang. *bean*), znacznik <c:set> może wykonać następujące rzeczy:

- ustawić właściwości ziarna;
- ustawić wartości mapy;
- tworzyć zmienne objęte zakresem na stronie, w żądaniu, w sesji lub w zakresie aplikacji.

W tabeli 3.10 opisano atrybuty akcji <c:set>.

Tabela 3.10. *Atrybuty <c:set>*

Nazwa	Typ	Opis
value	Object	Wyrażenia poddawane ewaluacji.
var	String	Nazwa eksportowanej zmiennej objętej zakresem, służącej do przechowywania wartości określonej w tej akcji.
scope	String	Zakres dla wartości var.
target	Object	Docelowy obiekt, którego właściwość zostanie ustawiona. Musi ewaluować do obiektu JavaBeans z metodą ustawiającą właściwość property lub do obiektu java.util.Map.
property	String	Nazwa właściwości, która ma być ustawiona w obiekcie docelowym.

Znacznik <c:set> służy do przeprowadzania następujących akcji:

- Ustawiania wartości zmiennej objętej zakresem w każdym zakresie JSP.
- Ustawiania właściwości określonego obiektu docelowego. Cel musi ewaluować do obiektu JavaBeans lub obiektu typu mapa.

Ustawianie w każdym zakresie JSP wartości zmiennej objętej zakresem

Jednym z zadań akcji <c:set> jest ustawienie zmiennych, które mogą być używane przez inne akcje na stronach.

Oto składnia:

```
<c:set value="wartość" var="nazwa_zmiennej" [scope="{page|request|session|application}"]/>
```

W listingu 3.36 przedstawiono przykład użycia znacznika `<c:set>`, aby ustawić wartość zmiennej `helloVar` objętej zakresem.

Listing 3.36. *Korzystanie ze znacznika <c:set>*

```
<c:set var="helloVar" value="witaj!" />
```

Kod z listingu 3.36 tworzy atrybut o nazwie `helloVar` z wartością `"witaj!"` w zakresie domyślnym, którym jest zakres strony. Można także tworzyć atrybuty w innym zakresie, np. w zakresie sesji. W tym przypadku należy określić zakres z akcją `<c:set>` za pomocą atrybutu `scope=""`. Kod z listingu 3.37 tworzy zmienną w zakresie sesji, używając znacznika `<c:set>`.

Listing 3.37. *Korzystanie z zakresu*

```
<c:set var="helloVar" value="witaj!" scope="session" />
```

Mogliśmy również wyrazić to poprzez dostarczanie wartości w akcji w zawartości sekcji `body`, tak jak przedstawiono w listingu 3.38.

Listing 3.38. *Korzystanie z akcji <c:set> z sekcją body*

```
<c:set var="helloVar" scope="session" >
witaj, świecie!
</c:set>
```

Wartość zmiennej może być także wyrażeniem EL, tak jak przedstawiono w listingu 3.39.

Listing 3.39. *Użycie wyrażenia EL przy ustawieniu wartości*

```
<c:set var="titleVar" value="${book.title}" scope="session" />
```

W listingu 3.39 właściwość `title` ziarna `book` jest ustawiona w zmiennej `titleVar`.
W listingu 3.40 przedstawiono odpowiednik skrypletowy znacznika `<c:set>` zilustrowanego w listingu 3.36.

Listing 3.40. *Odpowiednik skrypletowy znacznika <c:set>*

```
<%
String helloVar = "witaj!";
pageContext.setAttribute("helloVar ", helloVar);
%>
```

Ustawianie właściwości określonego obiektu docelowego

Aby móc ustawić właściwości ziarna lub wartości mapy za pomocą znacznika `<c:set>`, musimy użyć atrybutów `target` oraz `property` zamiast atrybutu `var`. Zdefiniują one ziarno oraz nazwę właściwości, które należy ustawić. Jeśli atrybutem `target` jest np. mapa, to atrybutem `property` jest nazwa klucza, a atrybut `value` jest wartością dla tego klucza.

Oto składnia:

```
<c:set value="wartość" target="obiekt_docelowy" property="nazwa_właściwości"/>
```

- Jeśli używany jest obiekt docelowy, cel musi ewaluować do obiektu JavaBeans lub `java.util.Map`.
- Jeśli celem jest obiekt JavaBeans, musi on zawierać odpowiednie metody pobierania lub ustawiania.

W listingu 3.41 przedstawiono, jak ustawić klucz mapy przy użyciu znacznika `<c:set>`.

Listing 3.41. *Ustawianie klucza mapy przy użyciu znacznika <c:set>*

```
<c:set target="bookMap" property="id" value="1">
```

Jest to odpowiednik wyrażenia `bookMap.put("id", "1");`. Można także dostarczyć wartość w sekcji `body` znacznika `<c:set>`.

W listingu 3.42 przedstawiono, jak ustawić właściwości ziarna, używając znacznika `<c:set>`.

Listing 3.42. *Ustawianie właściwości ziarna przy użyciu znacznika <c:set>*

```
<c:set target="book" property="book.title" value="Learning Java Web">
```

Kod z listingu 3.42 ustawia właściwość `title` dla ziarna o nazwie `book` jako `Learning Java Web`. Jest to odpowiednik `book.setTitle("Learning Java Web");`.

Akcja <c:remove>

Akcja `<c:remove>` usuwa zmienną z określonego zakresu. Zmienne ustawione za pomocą znacznika `<c:set>` w którymkolwiek z zakresów mogą być usunięte za pomocą akcji `<c:remove>` poprzez podanie nazwy zmiennej w atrybutach `var` i `scope`.

Oto składnia:

```
<c:remove var="nazwa_zmiennej" [scope="{page|request|session|application}"]/>
```

W tabeli 3.11 opisano atrybuty akcji `<c:remove>`.

Tabela 3.11. *Atrybuty akcji <c:remove>*

Nazwa	Typ	Opis
`var`	`String`	Nazwa objętej zakresem zmiennej, która ma być usunięta.
`scope`	`String`	Zakres dla zmiennej `var`.

W listingu 3.43 przedstawiono proste wykorzystanie akcji `<c:remove>`.

Listing 3.43. *Korzystanie z akcji <c:remove>*

```
<c:remove var=" helloVar " />
```

Akcja <c:catch>

Akcja `<c:catch>` zapewnia metodę przechwytywania wyjątków obiektu `java.lang.Throwable`, które są generowane przez wszelkie zagnieżdżone akcje. Ta akcja ma jeden atrybut zmiennej `var`, która posiada odniesienie do każdego wyjątku obiektu klasy `java.lang.Throwable`, występującego podczas realizacji dowolnej zagnieżdżonej akcji.

Oto składnia:

```
<c:catch [var="nazwa_zmiennej"]>
...zagnieżdżone akcje w sekcji body...
</c:catch>
```

W tabeli 3.12 opisano atrybut akcji `<c:catch>`.

Tabela 3.12. *Atrybuty akcji <c:catch>*

Nazwa	Typ	Opis
`var`	`String`	Nazwa eksportowanej zmiennej objętej zakresem dla wyjątku generowanego z zagnieżdżonej akcji.

Akcja `<c:catch>` może obsłużyć błędy pochodzące z dowolnej akcji przez zagnieżdżanie tych akcji w akcji `<c:catch>`. Gdy wygenerowany zostaje wyjątek, jest on przechowywany w zmiennej objętej zakresem strony, która jest identyfikowana przez atrybut zmiennej `var` znacznika. W listingu 3.44 przedstawiono użycie akcji `<c:catch>`.

Listing 3.44. *Korzystanie z akcji <c:catch>*

```
<body>

<c:catch var = "exception">
<% int i = 1/0;%>
</c:catch>

<c:if test = "${exception != null}">
<span> Wyjątek : ${exception}</span>

</c:if>
</body>
```

Oto dane wyjściowe:

```
Wyjątek : java.lang.ArithmeticException: / by zero
```

Tryby warunkowe

Znaczniki warunkowe dostarczane przez bibliotekę znaczników podstawowych JSTL stanowią alternatywę dla stosowania skrypletów w celu generowania zawartości dynamicznej na podstawie instrukcji warunkowych. W tabeli 3.13 zawarto opis akcji tego obszaru funkcjonalnego.

Tabela 3.13. *Warunkowe akcje w bibliotece znaczników podstawowych*

Akcja	Opis
`<c:if>`	Przeprowadza ewaluację własnej sekcji `body`, jeśli wyrażenie określone przez atrybut `test` jest ustawione jako `true`.
`<c:choose>`	Zapewnia kontekst dla realizacji warunków wzajemnie się wykluczających.
`<c:when>`	Stanowi alternatywę w ramach akcji `<c:choose>`.
`<c:otherwise>`	Stanowi ostatnią alternatywę w ramach akcji `<c:choose>`.

Akcja <c:if>

Akcja `<c:if>` służy do przetwarzania warunkowego i przeprowadza ewaluację wyrażenia, wyświetlając własną zawartość sekcji `body` wyłącznie, jeśli wyrażenie ewaluuje do wartości `true`.

Oto składnia bez zawartości sekcji `body`:

```
<c:if test="warunek_testowy" var="nazwa_zmiennej" [scope="{page|request|session|application}"]/>
```

Oto składnia z zawartością sekcji `body`:

```
<c:if test="warunek_testowy" [var="nazwa_zmiennej"] [scope="{page|request|session|application}"]>
. . . zawartość sekcji body . . .
</c:if>
```

Wyrażenie logiczne EL jest poddawane ewaluacji przy wykorzystaniu atrybutu `test`. Sekcja `body` danej akcji jest wykonywana wyłącznie wtedy, kiedy warunek testowy ewaluuje do wartości `true`. Wynik logiczny ewaluacji wyrażenia jest eksportowany za pomocą atrybutu `var` do zmiennej objętej zakresem. Domyślnym zakresem zmiennej `var` jest strona, ale przy użyciu atrybutu `scope` można ustawić dowolny z zakresów JSP.

W tabeli 3.14 przedstawiono atrybuty akcji `<c:if>`.

Tabela 3.14. *Atrybuty akcji <c:if>*

Nazwa	Typ	Opis
`test`	`Boolean`	Warunek testowy, który określa, czy zawartość sekcji `body` powinna być przetwarzana.
`var`	`String`	Nazwa eksportowanej zmiennej objętej zakresem dla otrzymanej wartości warunku testowego.
`scope`	`String`	Zakres dla zmiennej `var`.

W listingu 3.45 przedstawiono użycie akcji `<c:if>` z zawartością sekcji `body`.

Listing 3.45. *Korzystanie z akcji <c:if> z zawartością sekcji body*

```
<c:set var="number" value="9"/>
<c:if test="${ number < 10}" >
<c:out value ="liczba jest mniejsza niż 10"/>
</c:if>
```

Oto dane wyjściowe:

```
liczba jest mniejsza niż 10
```

Akcje <c:choose>, <c:when> oraz <c:otherwise>

Akcja `<c:choose>` pozwala obsłużyć warunki wzajemnie się wykluczające. Działa jak instrukcja Javy `switch` i pozwala wybrać między wieloma alternatywami. Używa akcji `<c:when>` (zamiast instrukcji `case`) oraz `<c:otherwise>` w celu dostarczenia domyślnej akcji, podobnie jak instrukcja `switch` postępuje z domyślną klauzulą.

Składnia akcji `<c:choose>` jest następująca:

```
<c:choose>
zawartość sekcji body (<c:when> oraz <c:otherwise>)
</c:choose>
```

Jak widać, akcja `<c:choose>` posiada dwie możliwe akcje zagnieżdżone, które tworzą jej sekcję `body`: `<c:when>` oraz `<c:otherwise>`. Składnia każdej z nich jest następująca:

```
<c:when test="warunek_testowy">
zawartość sekcji body
</c:when>
<c:otherwise>
zawartość sekcji body
</c:otherwise>
```

W tabeli 3.15 przedstawiono atrybuty akcji `<c:when>`.

Tabela 3.15. *Atrybuty akcji <c:when>*

Nazwa	Typ	Opis
`test`	`Boolean`	Warunek testowy, który określa, czy zawartość sekcji `body` powinna być przetwarzana.

W listingu 3.46 przedstawiono proste wykorzystanie akcji `<c:choose>`.

Listing 3.46. *Korzystanie z akcji <c:choose>*

```
<body>
<c:set var="number" value="10"/>
<c:choose>
<c:when test="${number < 10}">
    Liczba jest mniejsza niż 10.
</c:when>
<c:when test="${number > 10}">
    Liczba jest większa niż 10.
</c:when>
<c:otherwise>
    Liczba jest równa 10.
</c:otherwise>
</c:choose>
</body>
```

Oto dane wyjściowe:

```
Liczba jest równa 10.
```

Pętle i iteracje

Biblioteka JSTL oferuje dwie przydatne akcje dla pętli i iteracji: `<c:forEach>` w przypadku danych ogólnych oraz `<c:forTokens>` w przypadku łańcuchów znaków tokenizowanych.

W tabeli 3.16 przedstawiono akcje dla pętli i iteracji.

Tabela 3.16. *Akcje pętli i iteracji w bibliotece znaczników podstawowych*

Akcja	Opis
`<c:forEach>`	Przeprowadza iterację przez kolekcję obiektów.
`<c:forTokens>`	Przeprowadza iterację przez tokeny oddzielone dostarczonymi separatorami.

Akcja <c:forEach>

Akcja `<c:forEach>` przeprowadza iterację przez kolekcję obiektów.

Oto składnia iteracji przez kolekcję obiektów:

```
<c:forEach[var="nazwa_zmiennej"] items="kolekcja"
[varStatus="nazwa_stanu_zmiennej"]
[begin="początkowy_element"] [end="końcowy_element"] [step="wartość_inkrementacji"]>
. . . zawartość sekcji body . . . .
</c:forEach>
```

Oto składnia iteracji przeprowadzanej ustaloną liczbę razy:

```
<c:forEach [var="nazwa_zmiennej"]
[varStatus="nazwa_stanu_zmiennej"]
begin="początkowy_element" end="końcowy_element" [step="wartość_inkrementacji"]>
. . . zawartość sekcji body . . .
</c:forEach>
```

W tabeli 3.17 przedstawiono atrybuty akcji `<c:forEach>`.

Tabela 3.17. *Atrybuty <c:forEach>*

Nazwa	Typ	Opis
`var`	`String`	Nazwa eksportowanej zmiennej objętej zakresem dla danej pozycji w iteracji.
`items`	`"` tablice, kolekcja, enumeracja, iterator, mapa, łańcuch znaków	Kolekcja elementów do przeprowadzania iteracji.
`varStatus`	`String`	Nazwa eksportowanej zmiennej objętej zakresem w celu określenia stanu iteracji. Eksportowany obiekt jest typu `javax.servlet.jsp.jstl.core.LoopTagStatus`.
`begin`	`int`	Jeśli elementy są określone, iteracja rozpoczyna się od pozycji znajdującej się w określonym indeksie. Pierwszy element kolekcji posiada indeks 0. Jeśli elementy nie są określone, iteracja rozpoczyna się od indeksu ustawionego na określoną wartość.
`end`	`int`	Jeśli elementy są określone, iteracja kończy się na pozycji znajdującej się w określonym indeksie (włącznie). Jeśli elementy nie są określone, to iteracja kończy się, gdy indeks osiągnie określoną wartość.
`step`	`int`	Iteracja będzie przetwarzać elementy tylko co określoną wartość inkrementacji (np. co dwa elementy), począwszy od pierwszego.

Aby przeprowadzić iterację przez kolekcję obiektów, należy użyć następującej składni:

```
<c:forEach[var="nazwa_zmiennej"] items="kolekcja" [varStatus="nazwa_stanu_zmiennej"]
...zawartość sekcji body ....
</c:forEach>
```

Aby przeprowadzić iterację ustaloną liczbę razy, należy użyć następującej składni:

```
<c:forEach [var="nazwa_zmiennej"] [varStatus="nazwa_stanu_zmiennej"]
begin="początkowy_element" end="końcowy_element" [step="wartość_inkrementacji"]>
...zawartość sekcji body ...
</c:forEach>
```

W listingu 3.47 przedstawiono proste wykorzystanie akcji `<c:forEach>`.

Listing 3.47. *Korzystanie z akcji <c:forEach>*

```
<body>
<c:forEach var="i" begin="1" end="3">
Pozycja nr <c:out value="${i}"/><p>
</c:forEach>
</body>
```

Oto dane wyjściowe:

```
Pozycja nr 1
Pozycja nr 2
Pozycja nr 3
```

Akcja <c:forTokens>

Akcja `<c:forTokens>` przeprowadza iterację przez łańcuch tokenów oddzielonych zestawem separatorów.

Oto składnia:

```
<c:forTokens items="łańcuch_znaków_tokenów" delims="separatory"
[var="nazwa_zmiennej"] [varStatus="nazwa_stanu_zmiennej"]
[begin="początkowy_token"] [end="końcowy_token"] [step="wartość_inkrementacji"]>
. . . sekcja typu body . . .
</c:forTokens>
```

W tabeli 3.18 opisano wszystkie atrybuty `<c:forTokens>`.

Tabela 3.18. *Atrybuty <c:forTokens >*

Nazwa	Typ	Opis
`var`	`String`	Nazwa eksportowanej zmiennej objętej zakresem dla danej pozycji w iteracji.
`items`	`String`	Łańcuch tokenów przeznaczonych do iteracji.
`delims`	`String`	Zestaw separatorów (znaki, które oddzielają tokeny w łańcuchu).
`varStatus`	`String`	Nazwa eksportowanej zmiennej objętej zakresem, przeznaczonej do określenia stanu iteracji. Obiekt eksportowany jest typu `javax.servlet.jsp.jstl.core.LoopTagStatus`.
`begin`	`int`	Iteracja rozpoczyna się od tokenu znajdującego się w określonym indeksie. Pierwszy token posiada indeks `0`.
`end`	`int`	Iteracja kończy się na tokenie znajdującym się w określonym indeksie (włącznie).
`step`	`int`	Iteracja będzie przetwarzać tokeny tylko co określoną wartość inkrementacji (np. co dwa tokeny), począwszy od pierwszego.

Znacznik `<c:forTokens>` współpracuje z łańcuchem znaków oddzielonych separatorem. W listingu 3.48 przedstawiono wykorzystanie znacznika `<c:forTokens>`.

Listing 3.48. *Korzystanie z <c:forTokens>*

```
<body>
<c:forTokens items="Clojure,Groovy,Java, Scala" delims="," var="lang">
<c:out value="${lang}"/><p>
</c:forTokens>
</body>
```

Oto dane wyjściowe dla kodu z listingu 3.48:

```
Clojure
Groovy
Java
Scala
```

Akcje związane z adresem URL

Akcje związane z adresem URL służą do łączenia, importowania i przekierowywania w aplikacjach sieciowych. W tabeli 3.19 zawarto opis wszystkich akcji w podstawowej bibliotece związanych z adresem URL.

Tabela 3.19. *Akcje związane z adresem URL*

Akcja	Przeznaczenie
`<c:import>`	Importuje zawartość zasobu opierającego się na adresach URL.
`<c:param>`	Dodaje parametry żądania do adresu URL. Jest to akcja zagnieżdżona akcji `<c:import>`, `<c:url>` oraz `<c:redirect>`.
`<c:url>`	Tworzy adres URL z odpowiednio stosowanymi regułami przepisywania.
`<c:redirect>`	Wysyła przekierowania HTTP do klienta.

Rzućmy okiem na akcje związane z adresem URL.

Akcja <c:import>

Akcja `<c:import>` importuje zawartość zasobu opierającego się na adresie URL, zapewniając dodatkową funkcjonalność w stosunku do akcji `<jsp:include>`. Składnia akcji `<c:import>` jest następująca:

```
<c:import url ="adres_url [context="kontekst"] [charEncoding="kodowanie_znaków"]
[scope="application|page|request|session"] [var= "nazwa_zmiennej"] >
Opcjonalna zawartość sekcji body dla zagnieżdżonych znaczników <c:param>
</c:import>
```

- Jedynym wymaganym atrybutem jest `url`, który jest adresem URL zasobu do zaimportowania.
- Akcja `<c:param>`, wyjaśniona dalej, może być używana jako znacznik zagnieżdżony w zawartości sekcji `body` akcji `<c:import>`.

W tabeli 3.20 zawarto opis wszystkich atrybutów stosowanych w akcji `<c:import>`.

Tabela 3.20. *Atrybuty akcji <c:import>*

Nazwa	Typ	Opis
`url`	`String`	Adres URL zasobu do zaimportowania.
`context`	`String`	Nazwa kontekstu podczas uzyskiwania dostępu do względnego zasobu URL, który należy do kontekstu obcego.
`var`	`String`	Nazwa eksportowanej zmiennej objętej zakresem dla zawartości zasobu.
`scope`	`String`	Zakres dla zmiennej `var`.
`charEncoding`	`String`	Kodowanie znaków zawartości dla zasobu wprowadzania danych.
`varReader`	`String`	Nazwa eksportowanej zmiennej objętej zakresem dla zawartości zasobu.

W poprzednim rozdziale dowiedziałeś się, w jaki sposób akcja `<jsp:include>` pozwala hermetyzować funkcjonalność na jednej stronie JSP i umieścić ją na innej. Umożliwia np. dołączenie nagłówka i stopki, tak jak przedstawiono w listingu 3.49.

Listing 3.49. *Korzystanie z akcji <jsp:include>*

```
<body>
<jsp:include page='/WEB-INF/jsp/header.jsp'/>
<%-- zawartość --%>
<jsp:include page='/WEB-INF/jsp/footer.jsp'/>
</body>
```

Akcja `<jsp:include>` jest ograniczona do zawierania zasobu, który należy do tej samej aplikacji sieciowej co dołączona strona i jest określony jako względny adres URL.

Możesz użyć akcji <c:import> zamiast <jsp:include>, aby importować zasoby w tej samej aplikacji sieciowej. W listingu 3.50 przedstawiono, jak korzystać z akcji <c:import> zamiast <jsp:include>.

Listing 3.50. *Korzystanie z akcji <c:import>*

```
<body>
<c:import url='/WEB-INF/jsp/header.jsp'/>
<%-- zawartość --%>
<c:import url='/WEB-INF/jsp/footer.jsp'/>
</body>
```

Oprócz uzyskiwania dostępu do zasobów w tej samej aplikacji sieciowej możesz również wykorzystać akcję <c:import> do uzyskiwania dostępu do zasobów zewnętrznych lub zasobów w obcym kontekście. Aby uzyskać dostęp do zasobów zewnętrznych, należy określić bezwzględny adres atrybutu url. Aby uzyskać dostęp do zasobów w obcym kontekście, należy określić wartość atrybutu context, który opisuje ścieżkę kontekstową dla kontekstu obcego, oraz wartość atrybutu url, reprezentującego ścieżkę uwzględniającą kontekst w relacji do zasobu. W listingu 3.51 przedstawiono, jak korzystać z akcji <c:import>, aby zaimportować zasób z obcego kontekstu.

Listing 3.51. *Importowanie zasobów z kontekstu obcego*

```
<c:import url='/jsp/book.jsp' context='/kontekst_obcy'/>
```

■ **Uwaga** Atrybut charEncoding jest wymagany przy uzyskiwaniu dostępu do zasobów bezwzględnego adresu URL w sytuacji, kiedy nie jest obecny protokół HTTP i nie stosuje się kodowania ISO-8859-1.

Akcja <c:param>

Omówione w dalszej części rozdziału akcje <c:import>, <c:url> oraz <c:redirect> służą do obsługi adresów URL. Akcja <c:param> jest wykorzystywana do wprowadzania parametrów żądań i jest używana jako znacznik zagnieżdżony w sekcji body w akcjach <c:import>, <c:url> i <c:redirect>. Akcja <c:param> przeprowadza także kodowanie adresu URL.

Oto składnia z parametrem wartości określonym w atrybucie value.

```
<c:param name="nazwa_parametru" value="wartość"/>
```

Oto składnia z parametrem wartości określonym w zawartości sekcji body.

```
<c:param name="nazwa_parametru">
wartość parametru
</c:param>
```

W tabeli 3.21 opisano atrybuty akcji <c:param>.

Tabela 3.21. *Atrybuty akcji <c:param>*

Nazwa	Typ	Opis
name	String	Nazwa parametru łańcucha znaków kwerendy
value	String	Wartość parametru

W listingu 3.52 przedstawiono, w jaki sposób można użyć akcji <c:import> zamiast <jsp:include>. Można również określić parametry żądania dla załączonych plików za pomocą akcji <jsp:param>, tak jak przedstawiono w listingu 3.52.

Listing 3.52. *Korzystanie z akcji <jsp:param>*

```
<body>
<jsp:include page='/WEB-INF/jsp/company/companyHeader.jsp'>
<jsp:param name='user'
value='<%=session.getAttribute("userName")%>'/>
</jsp:include>
<%-- Zawartość strony pojawia się tutaj --%>
<jsp:include page='/WEB-INF/jsp/company/companyFooter.jsp'/>
</body>
```

W listingu 3.53 przedstawiono, w jaki sposób można wykorzystać akcję `<c:param>` zamiast `<jsp:param>`.

Listing 3.53. *Korzystanie z akcji <c:param>*

```
<body>
<c:import url='/WEB-INF/jsp/header.jsp'>
<c:param name='user'
value='${sessionScope.userName}'/>
</c:import>
<%-- zawartość sekcji typu body --%>
<c:import url='/WEB-INF/jsp/footer.jsp'/>
```

Akcja <c:url>

Akcja `<c:url>` tworzy adres URL z odpowiednio zastosowanymi regułami przepisywania. Może ona sformatować adres URL i zapisać go w zmiennej określonej przez atrybut `var`.

Oto składnia bez zawartości sekcji `body`:

```
<c:url value [context] [var] [scope]/>
```

Oto składnia z zawartością sekcji `body` do określenia parametrów łańcucha znaków kwerendy:

```
<c:url value [context] [var] [scope]>
<c:param> Akcje
</c:url>
```

- Wymagany jest tylko atrybut `value`, który jest adresem URL do przetworzenia.
- Podznaczniki `<c:param>` można również określić w sekcji `body` akcji `<c:url>` w celu dodania do adresu URL parametrów łańcucha kwerendy, które będą odpowiednio zakodowane, jeśli jest to konieczne.

Atrybuty akcji `<c:url>` zostały wymienione w tabeli 3.22.

Tabela 3.22. *Atrybuty akcji <c:url>*

Nazwa	Typ	Opis
`value`	`String`	Adresy URL, które mają być przetwarzane.
`context`	`String`	Nazwa kontekstu podczas określania względnego zasobu URL, który należy do obcego kontekstu.
`var`	`String`	Nazwa eksportowanej zmiennej objętej zakresem dla przetwarzanego adresu URL.
`scope`	`String`	Zakres dla zmiennej `var`.

W listingu 3.54 przedstawiono proste wykorzystanie akcji `<c:url>`.

Listing 3.54. *Korzystanie z akcji <c:url>*

```
<c:url var="homePage" scope="session" value="http://www.twojaksiegarnia.com" />
</body>
```

Kiedy dla atrybutu `value` określony jest adres URL względem kontekstu lub względem strony, akcja `<c:url>` poprzedza ścieżkę kontekstową aplikacji sieciowej do adresu URL. Jeśli np. ścieżką kontekstu aplikacji sieciowej jest */bookWeb/books*, akcja `<c:url value='/book.jsp'/>` wygeneruje adres URL: */bookWeb/books/book.jsp*.

Akcja <c:redirect>

Akcja przekierowania HTTP `<c:redirect>` wysyła do klienta przekierowanie przeglądarki do alternatywnego adresu URL. Akcja `<c:redirect>` zapewnia z przekierowaniem także przepisywanie adresu URL.

Oto składnia bez zawartości sekcji `body`:

```
<c:redirect url="wartość" [context="kontekst"]/>
```

Oto składnia z zawartością sekcji `body` w celu określenia parametrów łańcucha znaków kwerendy:

```
<c:redirect url="wartość" [context="kontekst"]>
<c:param> znaczniki podkategorii
</c:redirect>
```

Akcja `<c:redirect>` ma dwa atrybuty: adres URL, który będzie używany do przekierowania, oraz opcjonalny kontekst.

Atrybuty `<c:redirect>` zostały wymienione w tabeli 3.23.

Tabela 3.23. *Atrybuty akcji <c:redirect>*

Nazwa	Typ	Opis
`url`	`String`	Adres URL zasobu do przekierowania.
`context`	`String`	Nazwa kontekstu podczas przekierowania do względnego zasobu adresu URL, który należy do kontekstu obcego.

Adres URL, względny lub bezwzględny, ma taką samą regułę przepisywania URL jak akcja `<c:url>`. W listingu 3.55 przedstawiono przekierowanie do zewnętrznego zasobu, który jest bezwzględnym adresem URL.

Listing 3.55. *Korzystanie z akcji <c:redirect>*

```
<c:redirect url="http://www.twojaksiegarnia.com "/>
```

Przekierowania do zasobu w obcym kontekście można dokonać za pomocą atrybutu `context`. Określony adres URL musi zaczynać się od znaku / jako adres URL uwzględniający kontekst, a nazwa kontekstu musi również rozpoczynać się od znaku /. W listingu 3.56 przedstawiono przekierowanie do zasobu obcego w obcym kontekście.

Listing 3.56. *Przekierowanie do zasobu obcego w obcym kontekście.*

```
<c:redirect url="/obce_źródło.html" context="/obcy_kontekst" />
```

W tym rozdziale przyjrzeliśmy się praktycznemu zastosowaniu maszyny platformy Java EE na poziomie sieciowym. Omówione zostały komponenty sieciowe (serwlety i strony JSP), bogaty w funkcje język wyrażeń EL i gotowe do użycia niestandardowe akcje (biblioteka JSTL). Teraz omówimy najlepsze rozwiązania praktyczne z wykorzystaniem wzorców.

Najlepsze rozwiązania praktyczne: używanie wzorców

Podczas gdy platforma Java EE doskonale sprawdza się w kwestiach normalizacji infrastruktury enterprise, dostarczania modelu aplikacji oraz zapewniania składników odpowiednich do tworzenia aplikacji sieciowych, to interakcje bezpośrednie z komponentami platformy Java EE często skutkują ogromną ilością kodów dodatkowych, a nawet redundancją kodów. Korzystanie z platformy Java EE nie oznacza stosowania najlepszych praktyk tworzenia architektury oraz projektowania aplikacji. Dlatego Deepak Alur, John Crupi i Dan Malks jako pierwsi stworzyli wzorce projektowe platformy Java EE na podstawie własnego doświadczenia przy projektowaniu systemów enterprise. W tym podrozdziale dokonamy wprowadzenia do stworzonych przez Alura, Crupiego i Malksa wzorców Java EE na poziomie sieciowym.

■ **Uwaga** Książka *J2EE. Wzorce projektowe. Wydanie 2* autorstwa Alura, Crupiego oraz Malksa (Wydawnictwo Helion, 2004 r.) jest wysoce zalecana w celu poznania najlepszych praktyk tworzenia architektury i projektowania aplikacji. Wraz z platformami Java EE 6 oraz Java EE 7 zaszły jednak istotne zmiany we wzorcach biznesowych i persystencji Java EE. Niektóre wzorce, takie jak Lokalizator Usług (ang. *Service Locator*), zostały wycofane na rzecz wzorca Wstrzykiwanie Zależności (ang. *Dependency Injection*). Wzorce poziomu sieciowego pozostają jednak takie same. Można znaleźć obszerne omówienie nowych wzorców biznesowych i persystencji dla platformy Java EE w książce *Real World Java EE Patterns: Rethinking Best Practices* (wydanie drugie) autorstwa Adama Biena.

Poziom sieciowy hermetyzuje logikę prezentacji, która jest wymagana do świadczenia usług na rzecz klientów.

Poziom prezentacji ma następujące funkcje:

- Przechwytuje żądania klientów.
- Zapewnia funkcjonalności, takie jak uwierzytelnianie, autoryzacja, szyfrowanie, zarządzanie sesjami itd.
- Zapewnia dostęp do usług biznesowych.
- Konstruuje odpowiedź.
- Renderuje odpowiedzi do klientów.

Generalnie tworzenie aplikacji sieciowych wymaga rozwiązania typowego zestawu problemów.

- Gdy żądanie wpływa do aplikacji sieciowej, często musi być wstępnie przetworzone w celu zapewnienia określonych funkcjonalności, takich jak uwierzytelnianie, autoryzacja i szyfrowanie.
- Poziom prezentacji i logika biznesowa są często przemieszane. To sprawia, że poziom prezentacji jest trudny do utrzymania.
- Widoki są często kodowane z logiką nawigacji widoku. Powoduje to przemieszanie zawartości widoku i nawigacji widoku.
- Nie istnieje scentralizowany komponent do zarządzania widokiem, co skutkuje redundancją kodu i rozproszeniem kodu pomiędzy widokami.

Nie jest to pełna lista problemów, ale są to najbardziej typowe problemy występujące w aplikacji sieciowej. Na szczęście te problemy w aplikacji sieciowej można rozwiązać za pomocą wzorców poziomu sieciowego platformy Java EE. Sposób, w jaki należy korzystać z tych wzorców, różni się w zależności od problemu. W tabeli 3.24 opisano te wzorce.

■ **Uwaga** Dobra aplikacja typu enterprise składa się z wielu poziomów, a każdy z nich koncentruje się na własnych obowiązkach czy zagadnieniach, tak jak wyjaśniono w rozdziałach 1. i 2.

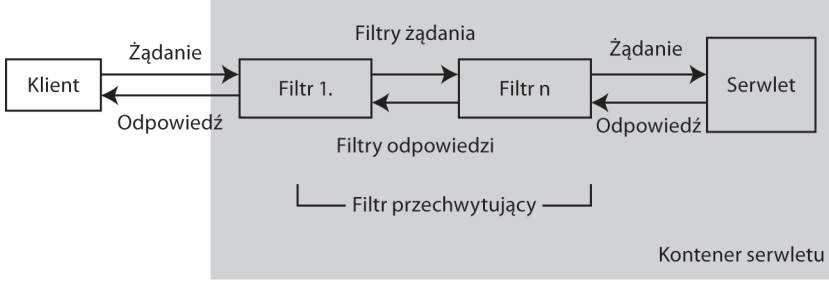
W kolejnych punktach przyjrzymy się każdemu z tych wzorców poziomu sieciowego platformy Java EE.

Tabela 3.24. *Wzorce poziomu prezentacji Java EE*

Wzorzec poziomu prezentacji	Opis
Filtr Przechwytujący (ang. *Intercepting Filter*)	Przetwarzanie wstępne i końcowe żądań oraz odpowiedzi klientów.
Kontroler Frontowy (ang. *Front Controller*)	Scentralizowany punkt dostępu do obsługi żądania poziomu prezentacji, który ma na celu wsparcie integracji usług systemowych, pobierania zawartości, zarządzania widokiem oraz nawigacji.
Pomocnik Widoku (ang. *View Helper*)	Hermetyzowanie logiki biznesowej w taki sposób, że nie jest ona przepleciona z logiką prezentacji.
Widok Kompozytowy (ang. *Composite View*)	Zarządzanie układem widoku niezależnie od zawartości.
Usługa Dla Pracownika (ang. *Service to Worker*)	Łączenie mikroframeworku kontrolera frontowego i pomocnika widoku z komponentem dyspozytora.
Widok Dyspozytora (ang. *Dispatcher View*)	Łączenie mikroframeworku kontrolera frontowego oraz pomocnika widoku z komponentem dyspozytora.

Filtr Przechwytujący

Aplikacja sieciowa otrzymuje różnego typu żądania, które wymagają pewnego rodzaju przetwarzania, np. w celu sprawdzenia, czy klient ma być uwierzytelniony przed rozpoczęciem nawigacji. Najlepszym sposobem zapewnienia mechanizmu przetwarzania żądań jest stosowanie komponentu przetwarzania zwanego **filtrem**. Filtry są stosowane, gdy trzeba zapewnić funkcjonalności wstępnego i końcowego przetwarzania żądań. Wzorzec Filtr Przechwytujący pojawił się wraz z wprowadzeniem filtrów do specyfikacji serwletów. Wzorzec ten wykorzystuje jeden filtr lub kilka filtrów podłączonych do bieżącej aplikacji w celu dostarczania usług takich jak uwierzytelnianie, kompresja, szyfrowanie oraz rejestrowanie. Na rysunku 3.8 przedstawiono wzorzec Filtr Przechwytujący.



Rysunek 3.8. *Wzorzec Filtr Przechwytujący*

Wzorzec Filtr Przechwytujący jest używany do przetwarzania wstępnego oraz przetwarzania końcowego żądań klientów, jak również odpowiedzi klientów. Odbywa się to poprzez przechwytywanie żądań i odpowiedzi. Filtry są podłączalne, w tym sensie, że można dodać je lub usunąć bez zmiany kodu.

Jeden z przypadków użycia, dla których wzorzec Filtr Przechwytujący jest uważany za najlepiej dopasowany, to sytuacja, gdy chcesz włączyć domyślnie w przeglądarce tryb dokumentu IE9. Przeglądarka Internet Explorer ma dwa tryby: tryb przeglądarki i tryb dokumentu. Przeglądarka zawsze wysyła dane dotyczące trybu przeglądarki do serwera, a serwer zawsze odpowiada danymi w trybie dokumentu. Dane trybu przeglądarki składają się z łańcucha znaków klienta użytkownika z informacjami o wersji silnika Trident, podczas gdy dane w trybie dokumentu składają się z metatagów określających tryb, w którym odpowiedź będzie renderowana w przeglądarce.

W listingu 3.57 przedstawiono prosty filtr odpowiedzi, który włącza domyślnie w przeglądarce tryb dokumentu IE9.

Listing 3.57. *Prosty filtr odpowiedzi*

```
1.package com.apress.filters
2.import javax.servlet.*;
3.import javax.servlet.http.HttpServletResponse;
4.import java.io.IOException;
5.import java.util.Enumeration;
6.
7./**
8. * Filtr do włączenia domyślnie trybu dokumentu IE9
9. *
10. */
11.public class ResponseHeaderFilter implements Filter {
12.     private FilterConfig filterConfig = null;
13.
14.     public void doFilter(ServletRequest aServletRequest, ServletResponse aServletResponse,
        ↪FilterChain chain)
15.             throws IOException, ServletException {
16.
17.         HttpServletResponse response = (HttpServletResponse) aServletResponse;
18.
19.         // konfiguracja parametrów dostarczonej odpowiedzi HTTP
20.         for (Enumeration e = filterConfig.getInitParameterNames(); e.hasMoreElements();) {
21.             String headerName = (String) e.nextElement();
22.             response.addHeader(headerName, filterConfig.getInitParameter(headerName));
23.         }
24.
25.         // przekazanie żądania/odpowiedzi
26.         chain.doFilter(aServletRequest, response);
27.     }
28.
29.     public void init(FilterConfig aFilterConfig) {
30.         filterConfig = aFilterConfig;
31.     }
32.
33.     public void destroy() {
34.         filterConfig = null;
35.     }
36.}
```

- **Wiersz 20.** — pobiera wszystkie parametry inicjacji z deskryptora wdrożenia.
- **Wiersz 22.** — dodaje te parametry do odpowiedzi.

W listingu 3.58 przedstawiono konfigurację filtra odpowiedzi w deskryptorze wdrożenia.

Listing 3.58. *Konfiguracja prostego filtra odpowiedzi*

```
1.<filter>
2.<filter-name>HTML5</filter-name>
3.<filter-class>com.apress.filters.ResponseHeaderFilter</filter-class>
4.<init-param>
5.<param-name>X-UA-Compatible</param-name>
6.<param-value>IE=edge,chrome=1</param-value>
7.</init-param>
8.</filter>
```

```
9.<filter-mapping>
10.<filter-name>HTML5</filter-name>
11.<url-pattern>/*</url-pattern>
12.</filter-mapping>
```

- **Wiersze 5. – 6.** — definiują parametry początkowe.

Kontroler Frontowy

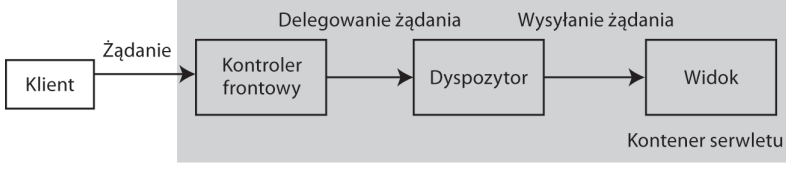
Aby aplikacje sieciowe były utrzymywalne, wszystkie żądania muszą przechodzić przez wspólny komponent scentralizowany. Brak scentralizowanego mechanizmu prowadzi do powstawania następujących problemów:

- Nie istnieje scentralizowany komponent do zarządzania widokiem, co skutkuje redundancją kodu i rozproszeniem kodu pomiędzy widokami.
- Widoki są często kodowane razem z logiką nawigacji widoku. Powoduje to przemieszanie zawartości widoku z nawigacją widoku.

Wzorzec Kontroler Frontowy zapewnia scentralizowany dostęp dla obsługi żądań w celu zapewnienia mechanizmów pobierania treści, zarządzania widokiem, nawigacji, walidacji, obsługi błędów, scentralizowanej kontroli bezpieczeństwa itd. Wzorzec Kontroler Frontowy najlepiej wdrażać w serwlecie. Korzystanie ze scentralizowanego serwletu do obsługi wszystkich żądań i odpowiedzi zapewnia następujące korzyści:

- Zapewnia pojedynczą lokalizację, z której odbywa się kontrola decyzji dotyczących uwierzytelniania i autoryzacji.
- Wszystkie adresy URL, które kontroler frontowy musi przetworzyć, mogą być mapowane na ten serwlet.
- Zapewnia scentralizowany punkt dostępu do obsługi zarządzania widokiem i nawigacji.
- Można zastosować wspólną logikę do kilku widoków.
- Zapewnia odseparowanie logiki prezentacji od logiki nawigacji i logiki biznesowej. Prowadzi to do powstania luźnych powiązań pomiędzy nimi.

Na rysunku 3.9 przedstawiono strukturę wzorca Kontroler Frontowy.



Rysunek 3.9. *Diagram klas wzorca Kontroler Frontowy*

Komponenty wzorca Kontroler Frontowy są następujące:

- kontroler (ang. *controller*),
- dyspozytor (ang. *dispatcher*),
- widok (ang. *view*).

Kontroler

Kontroler jest początkowym punktem obsługi żądania i koordynuje pracę z komponentem dyspozytora. Funkcje kontrolera obejmują:

- obsługę żądania, w tym wywoływanie usług bezpieczeństwa, takich jak uwierzytelnianie i autoryzacja;
- delegowanie zadań do usług biznesowych;
- obsługę błędów.

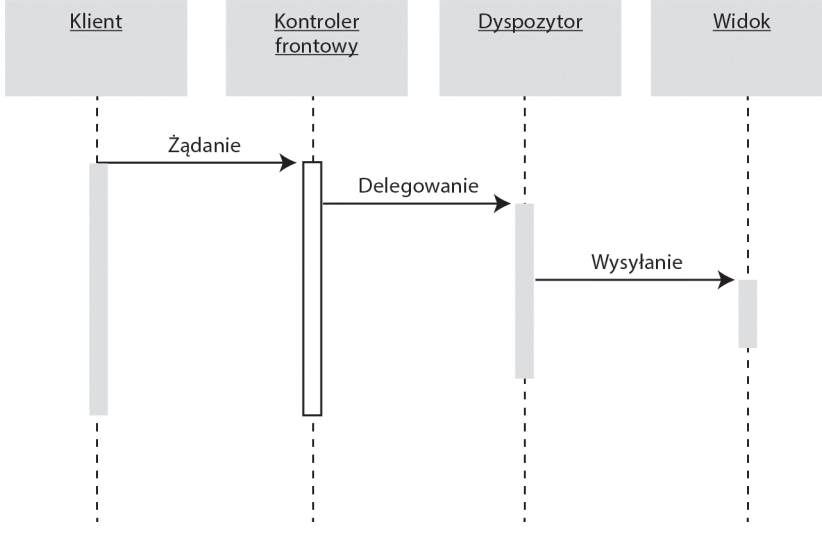
Dyspozytor

Dyspozytor jest odpowiedzialny za zarządzanie widokiem i nawigację.

Widok

Widok prezentuje i wyświetla informacje klientowi.

Na rysunku 3.10 przedstawiono diagram interakcji dla wzorca Kontroler Frontowy.



Rysunek 3.10. *Diagram interakcji wzorca Kontroler Frontowy*

Pomocnik Widoku

W aplikacji sieciowej zawartość prezentacji (czyli zawartość strony JSP) wymaga przetwarzania dynamicznej zawartości. Zmiany poziomu prezentacji pojawiają się często i są trudne do opracowania oraz utrzymania, gdy logika biznesowa i logika prezentacji są przemieszane. W istocie przeplatanie logiki biznesowej i logiki prezentacji sprawia, że prezentacja jest nieutrzymywalna. Wzorzec projektowy Pomocnik Widoku oddziela poziom prezentacji od poziomu biznesowego i generuje widok na podstawie szablonu. Komponenty wzorca Pomocnik Widoku są następujące:

- widok (ang. *view*),
- pomocnik widoku (ang. *view helper*).

Widok

Widok zawiera logikę formatowania prezentacji i deleguje logikę przetwarzania biznesowego w prezentacji do pomocnika.

Pomocnik widoku

Pomocnik widoku pełni następujące funkcje:

- Pomocnik widoku może być wdrożony jako implementacja obiektu JavaBeans lub niestandardowego znacznika. Użyliśmy tego wzorca w rozdziale 2.
- Pomocnicy służą jako model danych pośrednich widoku.

- Pomocnicy są odpowiedzialni za pobieranie danych z usługi biznesowej.
- Logika biznesowa prezentacji jest zhermetyzowana w pomocniku.

Na rysunku 3.11 przedstawiono strukturę wzorca Pomocnik Widoku.

Rysunek 3.11. *Diagram klas wzorca Pomocnik Widoku*

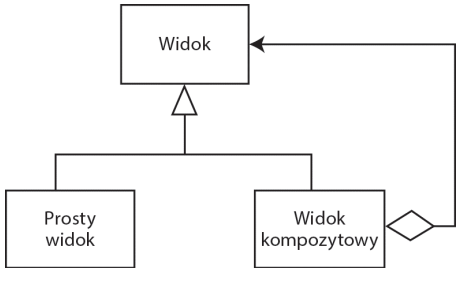
Widok Kompozytowy

W aplikacjach sieciowych widoki są często budowane przez osadzanie formatującego kodu bezpośrednio w każdym widoku. To sprawia, że modyfikowanie układu wielu widoków jest trudne. Wzorzec Widok Kompozytowy pozwala widokowi nadrzędnemu składać się z podwidoków. W związku z tym ogólny widok staje się kompozycją mniejszych podwidoków (ang. *subviews*), które są dołączane w nim dynamicznie.

Komponenty wzorca Widok Kompozytowy są następujące:

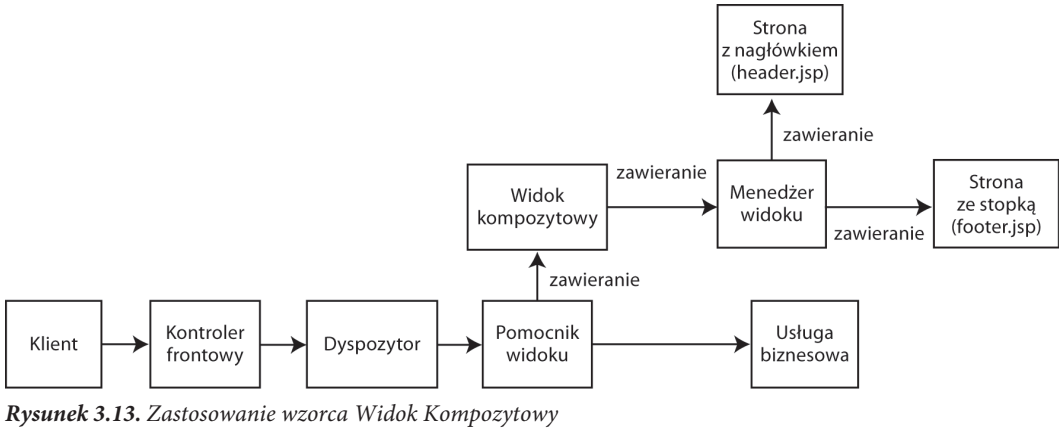
- **Widok podstawowy** (ang. *basic view*) jest podstawową abstrakcją widoku.
- **Widok kompozytowy** (ang. *composite view*) jest widokiem, który składa się z wielu podwidoków.
- **Widok** (ang. *view*) jest prostym widokiem, który nie posiada podwidoków.

Na rysunku 3.12 przedstawiono podstawową strukturę wzorca Widok Kompozytowy.



Rysunek 3.12. *Diagram klas wzorca Widok Kompozytowy*

Na rysunku 3.13 przedstawiono użycie wzorca Widok Kompozytowy.



Rysunek 3.13. *Zastosowanie wzorca Widok Kompozytowy*

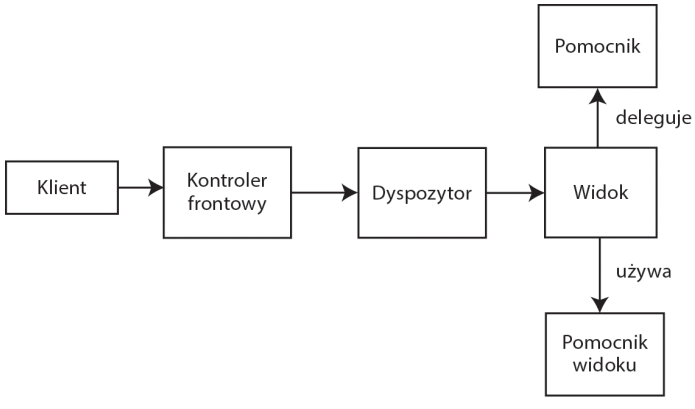
Widok Dyspozytora

Wzorzec projektowy Widok Dyspozytora łączy dwa wzorce, Kontroler Frontowy i Pomocnik Widoku, w jeden mikroframework z komponentem dyspozytora. Ma to na celu wykorzystanie zalet oferowanych przez każdy z tych wzorców.

We wzorcu Widok Dyspozytora dyspozytor jest odpowiedzialny za wybieranie i dostarczanie mechanizmu w celu statycznego lub dynamicznego przedstawiania kolejnego widoku użytkownikowi.

Komponenty wzorca Widok Dyspozytora są następujące (patrz rysunek 3.14):

- kontroler frontowy (ang. *front controller*),
- dyspozytor (ang. *dispatcher*),
- pomocnik (ang. *helper*),
- widok (ang. *view*),
- pomocnik widoku (ang. *view helper*).



Rysunek 3.14. *Diagram klas wzorca Widok Dyspozytora*

Kontroler frontowy

Kontroler frontowy pełni następujące funkcje:

- Centralizuje przetwarzanie żądań.
- Deleguje zadania do widoku za pomocą dyspozytora.
- Wykorzystuje pomocnika widoku do separacji zagadnień związanych z logiką biznesową od zagadnień związanych z logiką prezentacji.

Dyspozytor

Dyspozytor pełni następujące funkcje:

- Jest odpowiedzialny za zarządzanie widokiem oraz za nawigację.
- Deleguje zadania do widoku.
- Używa pomocnika w celu przesłania danych do widoku.

Pomocnik

Pomocnik pełni następujące funkcje:

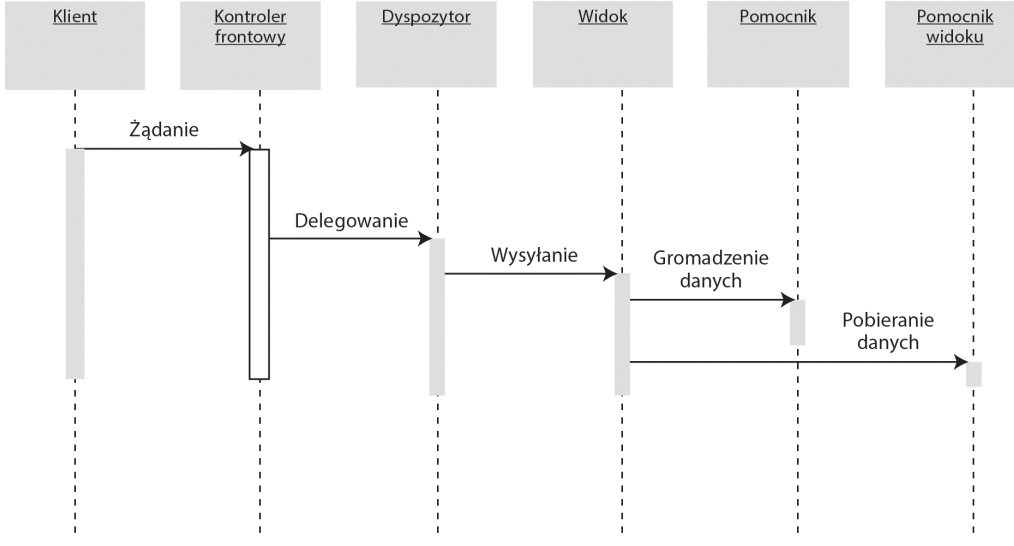
- Pomaga widokowi lub kontrolerowi w zakończeniu ich procesów przetwarzania.

Widok

Widok pełni następujące funkcje:

- Prezentuje i wyświetla informacje dla klienta.
- Pobiera dane ze źródła danych przy użyciu pomocnika widoku.

Na rysunku 3.15 przedstawiono diagram interakcji dla wzorca Widok Dyspozytora.



Rysunek 3.15. *Diagram interakcji dla wzorca Widok Dyspozytora*

Usługa Dla Pracownika

Wzorzec projektowy Usługa Dla Pracownika łączy dwa wzorce, Kontroler Frontowy i Pomocnik Widoku, w jeden mikroframework z komponentem dyspozytora. Ma to na celu wykorzystanie zalet oferowanych przez każdy z tych wzorców.

Wzorzec Usługa Dla Pracownika jest połączeniem dyspozytora z widokami i pomocnikami w celu zapewnienia obsługi żądań klientów oraz generowania dynamicznej zawartości jako odpowiedzi. Wzorzec Usługa Dla Pracownika może być zastosowany w przypadku, gdy aplikacja wymaga dynamicznego generowania zawartości.

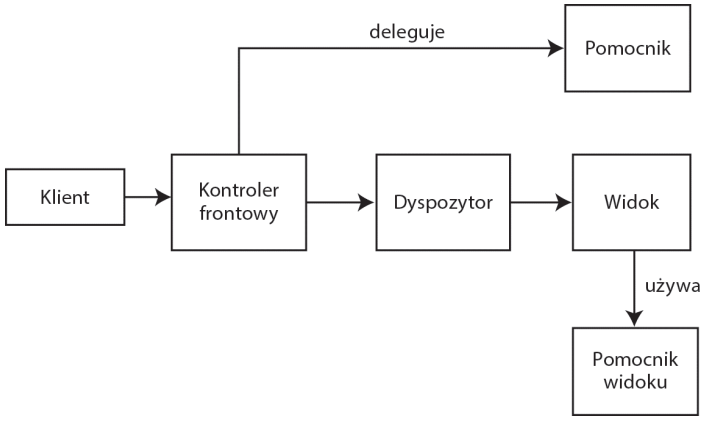
Komponenty wzorca Usługa Dla Pracownika są następujące (patrz rysunek 3.16):

- kontroler frontowy (ang. *front controller*),
- dyspozytor (ang. *dispatcher*),
- pomocnik (ang. *helper*),
- widok (ang. *view*),
- pomocnik widoku (ang. *view helper*).

Kontroler frontowy

Kontroler frontowy pełni następujące funkcje:

- Centralizuje przetwarzanie żądań.
- Deleguje zadania do widoku za pomocą dyspozytora.
- Używa pomocnika widoku do separacji zagadnień logiki biznesowej od zagadnień logiki prezentacji.



Rysunek 3.16. *Diagram klas wzorca Usługa Dla Pracownika*

Dyspozytor

Dyspozytor pełni następujące funkcje:

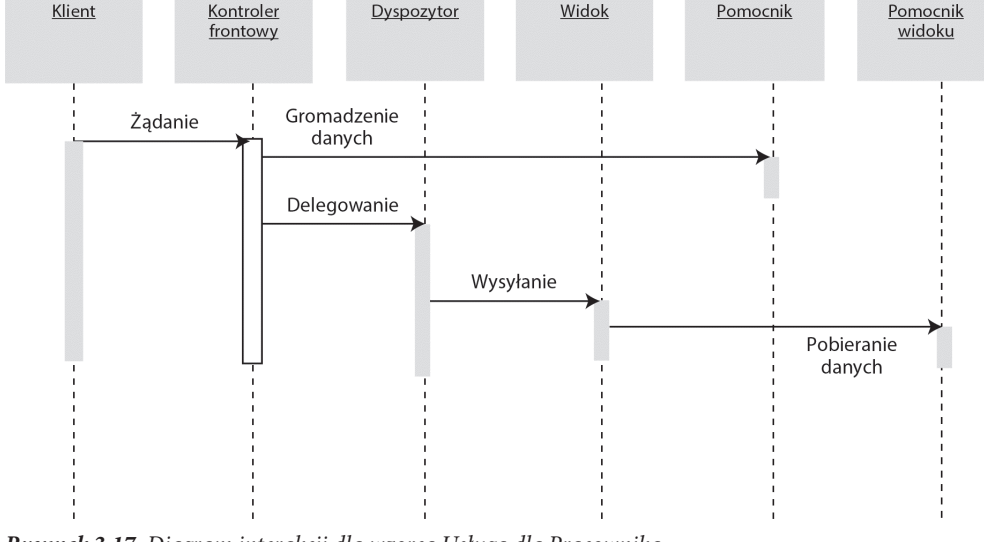
- Jest odpowiedzialny za zarządzanie widokiem i nawigację.
- Deleguje zadania do widoku.
- Używa pomocnika do wysyłania danych do widoku.

Pomocnik

Pomocnik pełni następujące funkcje:

- Pomaga widokowi lub kontrolerowi w zakończeniu ich procesów przetwarzania.
- Wysyła dane do widoku.

Na rysunku 3.17 przedstawiono diagram interakcji dla wzorca Usługa Dla Pracownika.



Rysunek 3.17. *Diagram interakcji dla wzorca Usługa dla Pracownika*

Wzorzec Usługa Dla Pracownika, podobnie jak wzorzec Widok Dyspozytora, jest połączeniem innych wzorców z katalogu. Oba te wzorce mikroframeworkowe opisują połączenie kontrolera i dyspozytora z widokami i pomocnikami. Różnica jest taka, że wzorzec Widok Dyspozytora opóźnia pobieranie treści aż do dynamicznego wygenerowania widoku, podczas gdy wzorzec Usługa Dla Pracownika pobiera zawartość w kontrolerze frontowym.

Omówione zostały właśnie wszystkie wzorce poziomu sieciowego platformy Java EE. Korzystając z tych wzorców, można rozwiązać problem domeny architektonicznej i problemy obszarów biznesowych. Należy jednak również zastosować **wzorce obiektowe** do rozwiązywania problemów domeny architektonicznej. Wiele wzorców projektowych może być stosowanych do zapewnienia możliwości wielokrotnego użytku, utrzymywalności i rozszerzalności. Można np. osiągnąć te trzy właściwości w aplikacji sieciowej za pomocą wzorców obiektowych, takich jak:

- Fabryka Abstrakcyjna (ang. *Abstract Factory*),
- Budowniczy (ang. *Builder*),
- Dekorator (ang. *Decorator*),
- Fasada (ang. *Facade*),
- Metoda Szablonowa (ang. *Template Method*).

Uwaga Pomimo stosowania wzorców Java EE i ewolucji przyrostowej niektóre aplikacje nie udają się z powodu błędnych decyzji architektonicznych. Te błędne decyzje architektoniczne zostały udokumentowane, tak aby nikt nie popełniał tych błędów ponownie. Są one nazywane antywzorcami (ang. *antipatterns*). Książka *J2EE Antipatterns* autorstwa Billa Dudneya, Stephena Asbury'ego, Josepha K. Krozaka oraz Kevina Wittkopfa (Wiley, 2003) jest doskonałym źródłem wiedzy na temat antywzorców.

Frameworki WWW Javy

Nauczyłeś się, że wzorzec MVC i wzorce poziomu sieciowego Java EE wspólnie ułatwiają tworzenie fundamentu architektonicznego dla budowania aplikacji sieciowych wielokrotnego użytku, utrzymywalnych oraz rozszerzalnych. Ponieważ projektanci zdobywają coraz więcej doświadczeń, zaczynają odkrywać obiekty ogólne, które mogą być używane wielokrotnie, przez co zaczynają się wyłaniać wzorce. Kiedy już zdobędziesz taką kolekcję obiektów ogólnych, zaczyna się wyłaniać framework. **Framework** jest zbiorem obiektów ogólnych i innych klas wspierających, które zapewniają infrastrukturę dla projektowania aplikacji. Frameworki są w istocie zbiorem wzorców projektowych strzeżonych przez podstawowe reguły frameworku, które omówimy poniżej. Framework Javy wykorzystuje dwa rodzaje wzorców.

- wzorce obiektowe (ang. *object-oriented* — OO),
- wzorce Java EE.

Framework wykorzystuje wzorce obiektowe do własnej konstrukcji w celu rozwiązywania **problemów domeny architektonicznej**, takich jak rozszerzalność, utrzymywalność, używanie wielokrotne, wydajność i skalowalność. Zarówno wzorce obiektowe, jak i wzorce Java EE rozwiązują **problemy domeny biznesowej**, takie jak przetwarzanie żądań, uwierzytelnianie, walidacja, zarządzanie sesjami, zarządzanie widokami itd. Frameworki rozwiązują te dwa główne obszary problemów architektonicznych i biznesowych za pomocą wzorców opartych na obiektach ogólnych i klasach wspierających, ściśle strzeżonych następującymi kluczowymi regułami:

- **Konfigurowalność**: jest zdolnością frameworku do korzystania z metadanych w celu zmiany zachowania się frameworku.
- **Odwrócenie sterowania**: w tradycyjnym stylu programowania kod domeny problemu steruje przepływem wykonywania aplikacji. Odwrócenie sterowania odnosi się do techniki, w której kod wielokrotnego użytku steruje wykonaniem kodu domeny problemu, w ten sposób kontrolując przepływ wykonywania aplikacji.

- **Luźne powiązania**: zasada ta odnosi się do niezależności współpracujących klas we frameworku, z których każda może być zamieniona bez wpływu na pozostałe klasy współpracujące.
- **Separacja zagadnień**: zasada ta odnosi się do potrzeby klasyfikacji obszarów domeny problemu i obsługi ich w odosobniony sposób tak, aby problemy jednego obszaru nie wpływały na problemy innego obszaru. Wielopoziomowa architektura platformy Java EE, która została omówiona w rozdziale 1., opiera się na regule separacji zagadnień.
- **Automatyzacja typowych funkcjonalności**: framework udostępnia mechanizmy do zautomatyzowanych rozwiązań dla podstawowych funkcjonalności domeny.

Dlaczego warto korzystać z frameworku?

Podczas gdy platforma Java EE doskonale sprawdza się w kwestiach normalizacji infrastruktury enterprise i zapewniania modelu aplikacji, istnieje kilka poważnych problemów związanych z tą technologią.

- Bezpośrednia interakcja z komponentami Java EE często skutkuje ogromną ilością kodu dodatkowego, a nawet redundancją kodu.
- Musisz napisać kod do obsługi typowych problemów domeny biznesowej.
- Musisz napisać kod do rozwiązywania problemów domeny architektonicznej.

Mógłbyś wdrożyć własny framework do rozwiązywania problemów związanych z tworzeniem aplikacji sieciowych opartych na Java EE z wykorzystaniem wzorców obiektowych oraz wzorców Java EE. Pisanie frameworku we własnym zakresie pociąga za sobą jednak nakłady pracy, które nie idą w parze z celami biznesowymi aplikacji. Ponadto jest mało prawdopodobne, aby framework tworzony we własnym zakresie mógł być modernizowany, a nowe wersje takiego frameworku mogą nigdy nie ujrzeć światła dziennego, w przeciwieństwie do frameworków głównego nurtu, które stale ewoluują, zamiast popadać w architekturalną entropię. Pamiętając o tym, przyjrzyjmy się teraz niektórym z dostępnych frameworków WWW opartych na maszynie JVM (patrz tabela 3.25). Tabela ta nie jest wyczerpująca. Dostępnych jest mnóstwo różnych frameworków, ale w tej książce omówimy najbardziej udane frameworki WWW oparte na maszynie JVM wymienione w tabeli.

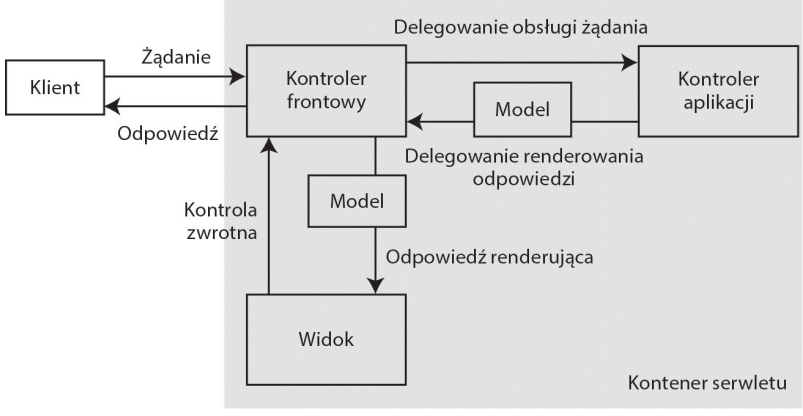
Tabela 3.25. *Frameworki WWW oparte na maszynie JVM*

Framework WWW	Kategoria	Język	Link do pobrania
Struts 2	Framework oparty na żądaniach	Java	*http://struts.apache.org/*
Spring Web MVC	Framework oparty na żądaniach	Java	*http://spring.io/tools*
JSF 2	Framework oparty na komponentach	Java	*http://www.oracle.com/technetwork/java/javaee/downloads/index.htm*
Grails 2	Framework szybkiego projektowania stron WWW	Groovy	*http://www.grails.org/download*
Play 2	Framework szybkiego projektowania stron WWW	Java i Scala	*http://www.playframework.com/download*

Wszystkie frameworki WWW wymienione w tabeli 3.25 opierają się na architekturze MVC Model-2, czyli na podstawowym wzorcu architektonicznym, który został omówiony w rozdziale 2. W typowej aplikacji sieciowej istnieje kilka zadań, które możesz chcieć wykonać dla każdego żądania przychodzącego, np. szyfrowanie. Pojedynczy kontroler aplikacji sieciowej pozwala scentralizować wszystkie zadania, które musi wykonać kontroler w aplikacji sieciowej, takie jak:

- centralizacja logiki do wysyłania żądań do widoków;
- sprawdzanie, czy użytkownik wysyłający żądanie wykonania określonej operacji posiada odpowiednią autoryzację.

Ponieważ aplikacje sieciowe oparte na wzorcu MVC posiadają coraz więcej widoków do wyświetlenia, kontroler w modelu aplikacji MVC staje się obiektem proceduralnym, który podejmuje zbyt wiele decyzji w celu zrenderowania tych widoków. Problem ten można rozwiązać poprzez zastosowanie kontrolera frontowego oraz metadanych konfiguracji zamiast użycia czystego wzorca architektury MVC. Takie rozwiązanie zostało przedstawione na rysunku 3.18.



Rysunek 3.18. *Wzorzec MVC z kontrolerem frontowym*

Gdy przychodzi żądanie HTTP od klienta, kontroler frontowy przeszukuje plik konfiguracji metadanych, aby wybrać właściwy kontroler aplikacji, który powinien obsłużyć dane żądanie HTTP. Kontroler aplikacji składa się z wywoływań logiki biznesowej i kontrolek logiki prezentacji dla żądań HTTP. Różnica między użyciem czystego wzorca MVC a zastosowaniem wzorca Kontroler Frontowy z wzorcem MVC jest taka, że serwlet kontrolera frontowego przeszukuje metadane konfiguracji, aby znaleźć kontroler aplikacji, który obsługuje przetwarzanie żądania HTTP, zamiast samemu decydować o wywołaniu określonych obiektów obsługi żądania HTTP.

W istocie jest to typowy wzorzec dla wielu wiodących frameworków WWW. Struts 2 oraz Spring Web MVC są przykładowymi frameworkami MVC opartymi na kontrolerze frontowym, który deleguje sterowanie do kontrolera aplikacji, wykorzystując w tym celu metadane konfiguracji przechowywane np. w pliku XML. We frameworku Struts 2 filtr `ServletFilter` jest przejawem kontrolera frontowego przedstawionego na rysunku 3.18. We frameworku JSF 2 jest to filtr `FacesServlet`, w Spring Web MVC `DispatcherServlet`, a w Grails 2 jest nim podklasa klasy `DispatcherServlet`.

Na tym kończy się rozdział. Od następnego rozdziału będziemy poruszali się po labiryncie frameworków WWW, które wykorzystują atuty platformy Java EE i maszyny JVM.

Podsumowanie

Niniejszy rozdział koncentrował się na zmieniających się technologiach i narzędziach wykorzystywanych w projekcie Java EE. Wprowadził Cię do typowych problemów związanych z tworzeniem aplikacji sieciowych i pokazał, jak można rozwiązać te problemy oraz jak sprawić — wykorzystując narzędzie dostępne w platformie Java EE (np. jednolity język wyrażeń EL) — aby aplikacja sieciowa stała się wielokrotnego użytku, utrzymywalna i rozszerzalna.

Język wyrażeń EL i biblioteka JSTL są dwoma doskonałymi implementacjami teorii separacji zagadnień biznesowych od zagadnień widoku. Mogą być one wykorzystywane do budowy aplikacji sieciowych wielokrotnego użytku, łatwych w utrzymaniu oraz bogatych w funkcjonalności. Ten rozdział zapewnia również wprowadzenie do wzorców poziomu sieciowego platformy Java EE oraz omawia, w jaki sposób wzorce Java EE dostarczają rozwiązań problemów powtarzających się w określonym kontekście. Następnie opisano, w jaki sposób frameworki rozwiązują problemy z obszarów architektonicznych oraz biznesowych poprzez dostarczanie obiektów ogólnych opartych na wzorcach i klasach wspierających. Na koniec przedstawiono, w jaki sposób framework WWW pozwala zrezygnować ze stosowania wzorców projektowych Java EE poprzez dostarczanie gotowych najlepszych rozwiązań praktycznych.

Skorowidz

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

Ż

PRZEWODNIK PO ŚWIECIE WWW W JĘZYKU JAVA!

Java
Projektowanie aplikacji WWW

Język Java zadebiutował na rynku blisko dwadzieścia lat temu. Od początku cieszy się ogromną popularnością i jest z powodzeniem używany do tworzenia projektów o różnym stopniu trudności. Język ten sprawdza się idealnie zarówno przy budowie zaawansowanych systemów bankowych, jak i prostych stron WWW. Przekonaj się, jaki potencjał tkwi w Javie!

Jeśli sięgniesz po tę książkę, błyskawicznie nauczysz się budować strony WWW przy użyciu serwletów i stron JSP. W kolejnych rozdziałach poznasz bardziej zaawansowane narzędzia — szkielet Struts 2, Spring Web MVC oraz JSF 2. Podczas tworzenia aplikacji działających w środowisku wirtualnej maszyny Javy nie jesteś skazany wyłącznie na ten język. Wiele problemów możesz rozwiązać sprawniej, za pomocą języków Scala lub Groovy. Książka ta jest doskonałą lekturą dla wszystkich osób chcących stworzyć stronę WWW w Javie i nie tylko!

Dzięki tej książce:

- skonfigurujesz swoje środowisko pracy
- zaznajomisz się z aktualnymi trendami w tworzeniu aplikacji
- poznasz popularne szkielety do tworzenia aplikacji WWW
- sprawdzisz możliwości języków Java, Groovy i Scala
- zbudujesz swoją stronę WWW

Apress

Nr katalogowy: 27247

Księgarnia internetowa: http://helion.pl

Zamówienia telefoniczne: 0 801 339900 / 0 601 339900

helion.pl księgarnia internetowa

Sprawdź najnowsze promocje: http://helion.pl/promocje
Książki najchętniej czytane: http://helion.pl/bestsellery
Zamów informacje o nowościach: http://helion.pl/nowosci

Helion

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

cena: 67,00 zł
ISBN 978-83-246-9806-6
9 788324 698066