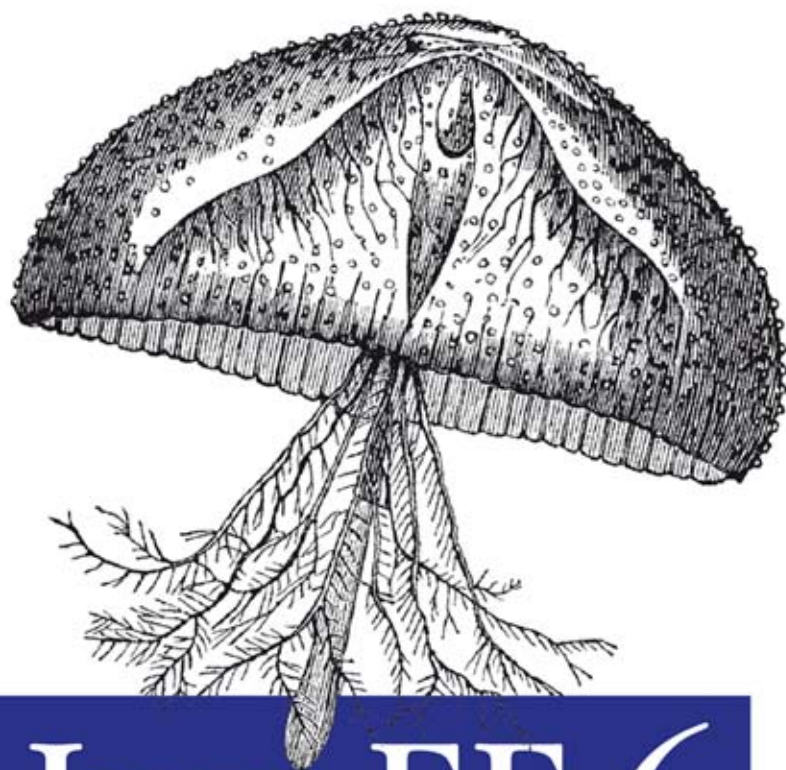


Podręczny przewodnik po JEE 6!



Java EE 6

Leksykon kieszonkowy



HELION

O'REILLY®

Arun Gupta

Tytuł oryginału: Java EE 6 Pocket Guide

Tłumaczenie: Mikołaj Szczepaniak

ISBN: 978-83-246-6640-9

© 2013 Helion S.A.

Authorized Polish translation of the English edition Java EE 6 Pocket Guide, ISBN 9781449336684, © 2012 Arun Gupta.

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

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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/jee61k>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Słowo wstępne	9
1. Java Platform, Enterprise Edition	12
Wprowadzenie	12
Elementy składowe	15
Co nowego w wersji Java EE 6	19
2. Komponenty zarządzane	23
Definiowanie i stosowanie komponentów zarządzanych	23
Wywołania zwrotne cyklu życia	25
3. Serwlety	26
Serwlety	26
Filtry serwletów	32
Obiekty nasłuchujące zdarzeń	33
Obsługa operacji asynchronicznych	37
Fragmenty konfiguracji	39
Bezpieczeństwo	40
Pakowanie zasobów	43
Odwzorowywanie błędów	44
Obsługa żądań wieloczęściowych	44
4. Java Persistence API	46
Encje	46
Jednostka utrwalania, kontekst utrwalania i menedżer encji	50
Tworzenie, odczytywanie, aktualizowanie i usuwanie encji	52
Sprawdzanie poprawności encji	56
Transakcje i blokowanie	58
Buforowanie	60

5. Enterprise JavaBeans	62
Stanowe komponenty sesyjne	62
Bezstanowe komponenty sesyjne	65
Singletonowe komponenty sesyjne	66
Komponenty sterowane komunikatami	68
Przenośne, globalne nazwy JNDI	70
Transakcje	71
Przetwarzanie asynchroniczne	74
Liczniki czasu	75
Interfejs Embeddable API	79
EJB.Lite	80
6. Konteksty i wstrzykiwanie zależności	82
Punkty wstrzykiwania	83
Kwalifikator i alternatywa	84
Producent i dyspozytor	86
Obiekty przechwytyjące i dekoratory	88
Zasięgi i konteksty	92
Stereotypy	94
Zdarzenia	95
Przenośne rozszerzenia	97
7. JavaServer Faces	100
Facelety	101
Obsługa zasobów	105
Komponenty złożone	105
Ajax	109
Żądanie HTTP GET	113
Punkty rozszerzeń serwera i klienta	115
Reguły nawigacji	118
8. Usługi sieciowe na bazie protokołu SOAP	119
Punkty końcowe usług sieciowych	121
Dynamiczne punkty końcowe na bazie interfejsu Provider	124
Punkty końcowe na bazie interfejsu Endpoint	125
Klient usługi sieciowej	127
Dynamiczny klient na bazie interfejsu Dispatch	129
Obiekty obsługujące	131

9. Usługi sieciowe zgodne ze stylem REST	134
Proste usługi sieciowe zgodne z REST	135
Wiązanie metod protokołu HTTP	137
Wiele reprezentacji jednego zasobu	140
Wiązanie żądania z zasobem	141
Odwzorowywanie wyjątków	143
10. Java Message Service	144
Wysyłanie wiadomości	147
Jakość usługi	149
Synchroniczne odbieranie komunikatu	150
Asynchroniczne odbieranie komunikatu	152
Tymczasowe miejsca docelowe	153
11. Bean Validation	154
Ograniczenia wbudowane	154
Definiowanie niestandardowych ograniczeń	158
Grupy sprawdzania poprawności	162
Integracja z technologią JPA	164
Integracja z frameworkiem JSF	166
A Programowanie i wdrażanie aplikacji platformy Java EE 6	
— pierwsze kroki	168
B Dodatkowe materiały	170
Specyfikacje technologii internetowych	170
Specyfikacje technologii korporacyjnych	170
Technologie usług sieciowych	171
Technologie zarządzania i zabezpieczeń	171
Skorowidz	173
O autorze	183
Kolofon	184

Rozdział 7. JavaServer Faces

Technologię JavaServer Faces (JSF) zdefiniowano w dokumencie JSR 314. Kompletną specyfikację tej technologii można pobrać ze strony internetowej <http://jcp.org/aboutJava/communityprocess/final/jsr314/index.html>.

JavaServer Faces jest frameworkiem interfejsu użytkownika działającym po stronie serwera i stworzonym z myślą o aplikacjach internetowych na bazie Javy. Framework JSF umożliwia między innymi:

- tworzenie stron internetowych przy użyciu zbioru komponentów interfejsu użytkownika wielokrotnego użytku, zgodnie ze wzorcem projektowym model-widok-kontroler (ang. *Model-View-Controller* — MVC);
- wiązanie komponentów z modelem po stronie serwera (dzięki temu jest możliwa dwukierunkowa migracja danych aplikacji i interfejsu użytkownika);
- obsługę nawigacji pomiędzy stronami w odpowiedzi na zdarzenia interfejsu użytkownika i interakcje z modelem;
- zarządzanie stanem komponentów interfejsu użytkownika pomiędzy żądaniami serwera;
- udostępnianie prostego modelu kojarzenia zdarzeń generowanych przez klienta z kodem wykonywanym po stronie aplikacji;
- łatwą budowę i wielokrotne stosowanie niestandardowych komponentów interfejsu użytkownika.

Aplikacja JSF składa się z następujących elementów:

- zbioru stron internetowych, na których umieszczono komponenty interfejsu użytkownika;
- zbioru komponentów zarządzanych; jeden zbiór takich komponentów wiąże komponenty interfejsu użytkownika z modelem po stronie serwera (który zwykle składa się z komponentów CDI lub komponentów zarządzanych), drugi zbiór pełni funkcję kontrolera (na kontroler składają się zwykle komponenty EJB lub CDI);
- opcjonalnego deskryptora wdrożenia (pliku *web.xml*);
- opcjonalnego pliku konfiguracyjnego *faces-config.xml*;
- opcjonalnego zbioru obiektów niestandardowych, jak konwertery czy obiekty nasłuchujące, utworzonych przez programistę aplikacji.

Facelety

Facelety to **język deklarowania widoków** (nazywany też mechanizmem obsługi widoków) stworzony z myślą o frameworku JSF. Facelety mają zastąpić technologię JSP, której obsługę zachowano wyłącznie z myślą o zapewnieniu zgodności wstecz. Nowe elementy wprowadzone w drugiej wersji specyfikacji JSF, w tym komponenty złożone i obsługa technologii Ajax, są dostępne tylko dla autorów stron stosujących facelety. Do najważniejszych zalet faceletów należą rozbudowany system szablonów, łatwość tworzenia i możliwość wielokrotnego stosowania komponentów, lepszy system raportowania o błędach (z uwzględnieniem numerów wierszy) oraz struktura stworzona z myślą o wygodzie projektantów.

Strony faceletów są tworzone przy użyciu znaczników języka XHTML 1.0 i elementów kaskadowych arkuszy stylów (ang. *Cascading Style Sheets* — CSS). Dokument XHTML 1.0 jest tłumaczony na język HTML 4 zgodnie z regułami standardu XML 1.0. Strona musi być zgodna ze specyfikacją XHTML-1.0-Transitional DTD (patrz strona http://www.w3.org/TR/xhtml1/#a_dtd_XHTML-1.0-Transitional).

Proste strony faceletów można definiować, stosując elementy języka XHTML:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Tytuł strony zbudowanej przy użyciu faceletów</title>
  </h:head>
  <h:body>
    Witaj w świecie faceletów.
  </h:body>
</html>
```

W powyższym kodzie po prologu XML-a następuje deklaracja typu dokumentu (DTD). Głównym elementem strony jest `html`, czyli element należący do przestrzeni nazw <http://www.w3.org/1999/xhtml>. Przestrzeń nazw języka XML jest deklarowana dla konkretnej biblioteki znaczników używanej w kodzie danej strony. Komponenty można dodawać za pomocą znaczników faceletów (rozpoczynających się od przedrostka `h:`) oraz za pomocą znaczników języka HTML.

Standardowy zbiór bibliotek znaczników obsługiwanych przez facelety opisano w tabeli 7.1.

Tabela 7.1. Standardowy zbiór bibliotek znaczników obsługiwanych przez facelety

Przedrostek	URI	Przykłady
h	http://java.sun.com/jsf/html	h:head, h:inputText
f	http://java.sun.com/jsf/core	f:facet, f:actionListener
c	http://java.sun.com/jsp/jstl/core	c:forEach, c:if
fn	http://java.sun.com/jsp/jstl/functions	fn:toUpperCase, fn:contains
ui	http://java.sun.com/jsf/facelets	ui:component, ui:insert

Konwencja nakazuje stosowanie rozszerzenia *.xhtml* dla stron internetowych tworzonych w języku XHTML.

Facelety oferują integrację z językiem wyrażeń EL (od ang. *Expression Language*). Dzięki temu możemy stosować dwukierunkowe powiązania łączące wewnętrzne komponenty z interfejsem użytkownika aplikacji:

```
Witaj w świecie faceletów, mam na imię #{name.value}!
```

W powyższym kodzie `#{name}` jest wyrażeniem języka EL odwołującym się do pola komponentu CDI o zasięgu żądania:

```
@Named
@RequestScoped
public class Name {
    private String value;
    // ...
}
```

W komponencie CDI należy użyć adnotacji `@Named`, aby umożliwić wstrzykiwanie tego komponentu do wyrażeń języka EL. Zaleca się stosowanie komponentów zgodnych ze specyfikacją CDI zamiast komponentów oznaczonych adnotacją `@javax.faces.bean.ManagedBean`.

Podobnie, do wyrażenia języka EL można wstrzyknąć komponent EJB:

```
@Stateless
@Named
public class CustomerSessionBean {
    public List<Name> getCustomerNames() {
        // ...
    }
}
```

W powyższym kodzie zdefiniowano bezstanowy komponent sesyjny z pojedynczą metodą biznesową zwracającą listę nazwisk klientów. Adnotacja `@Named` oznacza ten komponent jako przeznaczony do wstrzyknięcia w wyrażeniu języka EL. Komponent można teraz zastosować w wyrażeniu EL faceletów:

```
<h:dataTable value="#{customerSessionBean.customerNames}" var="c">
    <h:column>#{c.value}</h:column>
</h:dataTable>
```

W powyższym kodzie zwrócona lista nazwisk klientów jest wyświetlana w tabeli. Warto zwrócić uwagę na metodę `getCustomerNames`, która jest dostępna jako właściwość w wyrażeniu języka EL.

Facelety oferują też mechanizmy sprawdzania poprawności wyrażen EL w czasie kompilacji.

Co więcej, facelety oferują rozbudowany system szablonów, który umożliwia zachowywanie spójnego wyglądu i działania wszystkich stron aplikacji internetowej. Stronę bazową (określaną mianem **szablonu**) można utworzyć za pomocą znaczników szablonowych faceletów. Strona bazowa definiuje domyślną strukturę właściwych stron, w tym miejsca dla treści, która będzie definiowana przy użyciu danego szablonu. **Strony klienckie**, które korzystają z tego szablonu, przekazują właściwą treść do miejsc zdefiniowanych przez ten szablon.

W tabeli 7.2 opisano kilka najczęściej stosowanych znaczników szablonów i stron klienckich szablonów.

Tabela 7.2. Popularne znaczniki faceletów i szablonów

Znacznik	Opis
<code>ui:composition</code>	Definiuje układ strony, który może obejmować szablon. W przypadku zastosowania atrybutu <code>template</code> znaczniki potomne tego znacznika definiują układ szablonu. W przeciwnym razie znacznik zawiera grupę elementów, czyli strukturę złożoną, którą można wstawić w dowolnym miejscu kodu. Treść spoza tego znacznika jest ignorowana.
<code>ui:insert</code>	Znacznik stosowany na stronach szablonowych służy do definiowania obszaru wstawiania właściwej treści w szablonie. Odpowiedni znacznik <code>ui:define</code> w kodzie strony klienckiej szablonu umożliwia zastąpienie treści.
<code>ui:define</code>	Znacznik używany w kodzie strony klienckiej szablonu. Definiuje treść wstawianą w miejscu wyznaczonym przez szablon (za pomocą odpowiedniego znacznika <code>ui:insert</code>).
<code>ui:component</code>	Umieszcza w drzewie komponentów JSF nowy komponent interfejsu użytkownika. Komponenty i fragmenty treści spoza tego znacznika są ignorowane.
<code>ui:fragment</code>	Znaczenie tego znacznika jest podobne jak w przypadku <code>ui:component</code> , tyle że treść spoza tego znacznika nie jest ignorowana.
<code>ui:include</code>	Dołącza dokument wskazywany przez atrybut <code>src</code> (dołączony dokument staje się częścią bieżącej strony na bazie faceletów).

Strona szablonu może mieć następującą postać:

```
<h:body>

    <div id="top">
        <ui:insert name="top">
            <h1>Faceletry są super!</h1>
        </ui:insert>
    </div>

    <div id="content" class="center_content">
        <ui:insert name="content">Treść</ui:insert>
    </div>

    <div id="bottom">
        <ui:insert name="bottom">
            <center>Powered by GlassFish</center>
        </ui:insert>
    </div>

</h:body>
```

W powyższym kodzie struktura została zdefiniowana za pomocą elementu `<div>` i arkusza stylów CSS (listing nie zawiera definicji stylów). Znacznik `ui:insert` definiuje treść, która zostanie zastąpiona przez stronę kliencką szablonu.

Poniżej pokazano kod strony klienckiej tego szablonu:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">

    <body>

        <ui:composition template="./template.xhtml">

            <ui:define name="content">
                <h:dataTable
                    value="#{customerSessionBean.customerNames}"
                    var="c">
                    <h:column>#{c.value}</h:column>
                </h:dataTable>
            </ui:define>

        </ui:composition>

    </body>
</html>
```

W powyższym kodzie nie zdefiniowano znaczników `ui:insert` nazwanych `top` i `bottom`, zatem w odpowiednich sekcjach zostaną użyte zapisy strony szablonowej. Strona kliencka zawiera element `ui:define` z nazwą pasującą do elementu `ui:insert` w szablonie, zatem w tym przypadku treść zostanie prawidłowo zastąpiona.

Obsługa zasobów

Framework JSF definiuje standardowy mechanizm obsługi takich zasobów jak obrazy, arkusze stylów CSS czy skrypty języka JavaScript. Wymienione zasoby są niezbędne do prawidłowego wyświetlania komponentów.

Zasoby tego typu można umieszczać w katalogu */resources* aplikacji internetowej lub w katalogu *META-INF/resources* w ścieżce do klas. Zasoby można dzielić według wersji językowych i według numerów wydań oraz gromadzić w większych bibliotekach.

Odwołanie do zasobu można zapisać w formie wyrażenia języka EL:

```
<a href="#{resource['header.jpg']}">kliknij tutaj</a>
```

W tym przykładzie plik graficzny *header.jpg* został umieszczony w standardowym katalogu zasobów.

Jeśli zasób znajduje się w folderze *corp* (czyli w folderze zasobów biblioteki), dostęp do tego zasobu można uzyskać za pomocą atrybutu *library*:

```
<h:graphicImage library="corp" name="header.jpg" />
```

Skrypt języka JavaScript można dołączyć w następujący sposób:

```
<h:outputScript
  name="myScript.js"
  library="scripts"
  target="head"/>
```

W tym przykładzie plik *myScript.js* jest zasobem języka JavaScript umieszczonym w katalogu *scripts* w standardowym folderze zasobów.

Arkusz stylów CSS można dołączyć w następujący sposób:

```
<h:outputStylesheet name="myCSS.css" library="css" />
```

Interfejs *API ResourceHandler* dodatkowo oferuje programowe mechanizmy obsługi zasobów.

Komponenty złożone

Framework JSF definiuje komponent złożony jako taki, który składa się z co najmniej jednego komponentu JSF opisanego w pliku znaczników faceletów. Odpowiedni plik *.xhtml* należy umieścić w bibliotece zasobów. Takie rozwiązanie umożliwia tworzenie komponentów wielokrotnego użytku na podstawie dowolnego fragmentu strony.

Definiowanie komponentu kompozytowego polega na **definiowaniu strony**, natomiast stosowanie tego komponentu sprowadza się do **stosowania strony**. Strona definiująca komponent zawiera metadane (parametry) opisane w znaczniku `<cc:interface>` oraz właściwą implementację w znaczniku `<cc:implementation>`, gdzie `cc` jest przedrostkiem dla przestrzeni nazw <http://java.sun.com/jsf/composite/>. Wymagania dotyczące definiowania metadanych być może zostaną złagodzone w przyszłych wersjach specyfikacji JSF — niezbędne informacje będzie wówczas można umieszczać w samej implementacji.

Komponent kompozytowy można zdefiniować także przy użyciu elementów frameworku JSF 1.2, jednak budowa komponentów w ten sposób wymaga dobrej znajomości cyklu życia JSF i odpowiedniego przygotowania wielu plików. Framework JSF2 znacznie upraszcza konstruowanie komponentów złożonych wyłącznie na poziomie plików języka XHTML.

Przypuśćmy, że przykładowy facelet zawiera następujący fragment kodu odpowiedzialny za wyświetlanie formularza logowania:

```
<h:form>
  <h:panelGrid columns="3">
    <h:outputText value="Nazwa:" />
    <h:inputText value="#{user.name}" id="name"/>
    <h:message for="name" style="color: red" />
    <h:outputText value="Hasło:" />
    <h:inputText value="#{user.password}"
      id="password"/>
    <h:message for="password" style="color: red" />
  </h:panelGrid>

  <h:commandButton actionListener="#{userService.register}"
    id="loginButton"
    action="status"
    value="Wyślij"/>
</h:form>
```

Powyższy kod wyświetla tabelę złożoną z dwóch wierszy i trzech kolumn (patrz rysunek 7.1).



Rysunek 7.1. Strona faceletów JSF wyświetlona w oknie przeglądarki internetowej

Pierwsza kolumna zawiera etykiety pól formularza; druga kolumna zawiera pola tekstowe, w których użytkownik powinien wpisać dane uwierzytelniające. Trzecia kolumna (która początkowo jest pusta) służy do wyświetlania komunikatów związanych z wartościami w odpowiednich polach. Pierwszy wiersz kojarzy pole tekstowe formularza z polem `User.name`; drugi wiersz kojarzy pole tekstowe formularza z polem `User.password`. Formularz zawiera też przycisk polecenia, którego kliknięcie powoduje wywołanie metody `register` komponentu `UserService`.

Tak skonstruowany formularz logowania można wyświetlać na wielu stronach. Okazuje się jednak, że nie musimy za każdym razem powtarzać tego kodu — wystarczy przekształcić ten fragment w komponent złożony. W tym celu interesujący nas fragment należy skopiować do pliku *.xhtml*, a sam plik należy skopiować do biblioteki w standardowym katalogu zasobów. Zgodnie z zasadą konwencji ponad konfiguracją do wspomnianego fragmentu automatycznie zostanie przypisana przestrzeń nazw i nazwa znacznika.

Jeśli pokazany wcześniej fragment zostanie skopiowany do pliku *login.xhtml* w katalogu *resources/mycomp*, strona definiująca będzie miała następującą postać:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:cc="http://java.sun.com/jsf/composite"
  xmlns:h="http://java.sun.com/jsf/html">

  <!-- INTERFEJS -->
  <cc:interface>
  </cc:interface>

  <!-- IMPLEMENTACJA -->
  <cc:implementation>
    <h:form>
      <h:panelGrid columns="3">
        <h:outputText value="Nazwa:" />
        <h:inputText value="#{user.name}" id="name"/>

        <!-- ... -->

      </h:panelGrid>
    </h:form>
  </cc:implementation>
</html>
```

W tym kodzie element `cc:interface` definiuje metadane, które z kolei opisują cechy komponentu (w tym atrybuty, fasety, punkty wiązania z metodami nasłuchującymi zdarzeń). Element `cc:implementation` zawiera kod języka znaczników wstawiany w miejsce komponentu złożonego.

Przestrzeń nazw komponentu złożonego jest konstruowana przez konkatencję adresu `http://java.sun.com/jsf/composite/` i nazwy `mycomp`. Nazwa znacznika jest tworzona na podstawie nazwy pliku strony (bez rozszerzenia `.xhtml`):

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:mc="http://java.sun.com/jsf/composite/mycomp"

  <!-- ... -->
  <mc:login/>

</html>
```

Przypuśćmy, że fragment kodu musi przekazywać różne wyrażenia (zamiast stosowanego wcześniej wyrażenia `#{user.name}`) i wywoływać różne metody (zamiast wywoływanej do tej pory metody `#{userService.register}`) w zależności o strony, na której umieszczono komponent złożony. Odpowiednie wartości mogą być przekazywane przez stronę definiującą:

```
<!-- INTERFEJS -->
<cc:interface>
  <cc:attribute name="name"/>
  <cc:attribute name="password"/>
  <cc:attribute name="actionListener"
    method-signature=
      "void action(javax.faces.event.Event)"
    targets="ccForm:loginButton"/>
</cc:interface>

<!-- IMPLEMENTACJA -->
<cc:implementation>
  <h:form id="ccForm">
    <h:panelGrid columns="3">
      <h:outputText value="Nazwa:" />
      <h:inputText value="#{cc.attrs.name}" id="name"/>
      <h:message for="name" style="color: red" />
      <h:outputText value="Hasło:" />
      <h:inputText value="#{cc.attrs.password}"
        id="password"/>
      <h:message for="password" style="color: red" />
    </h:panelGrid>

    <h:commandButton id="loginButton"
      action="status"
      value="submit"/>
  </h:form>
</cc:implementation>
```

W tym kodzie wszystkie parametry dla jasności wymieniono w elemencie `cc:interface`. Trzeci parametr zawiera atrybut odwołujący się do pola `ccForm:loginButton`.

Najważniejsze cechy elementu `cc:implementation`:

- Element `h:form` zawiera atrybut `id`. Atrybut `id` jest niezbędny, jeśli przycisk formularza ma być przedmiotem bezpośrednich odwołań.
- Element `h:inputText` używa teraz wyrażenia `{cc.attrs.xxx}` (zamiast stosowanego wcześniej wyrażenia `{user.xxx}`). `{cc.attrs}` jest domyślnym wyrażeniem języka EL stosowanym przez autorów komponentów kompozytowych i umożliwiającym dostęp do atrybutów bieżącego komponentu. W tym przypadku `{cc.attrs}` zapewnia dostęp do atrybutów `name` i `password`.
- `actionListener` jest punktem dowiązania metody nasłuchującej zdarzeń. W tym przypadku zastosowano atrybut `method-signature` opisujący sygnaturę odpowiedniej metody.
- Element `h:commandButton` zawiera atrybut `id`, dzięki czemu może być jednoznacznie identyfikowany w ramach elementu `h:form`.

W kodzie strony korzystającej z komponentu wartości `user`, `password` i `actionListener` są przekazywane w formie wymaganych atrybutów:

```
<ez:login
    name="{user.name}"
    password="{user.password}"
    actionListener="{userService.register}"/>
```

Strona korzystająca z tego komponentu może teraz przekazywać różne komponenty wewnętrzne, a zdarzenie naciśnięcie przycisku wysłania formularza będzie powodowało wywołania różnych metod biznesowych.

W największym skrócie komponenty złożone mają następujące zalety:

- Koncepcja tych komponentów jest w pełni zgodna z zasadą unikania powtórzeń (DRY), ponieważ umożliwia umieszczenie w jednym miejscu (pliku) kodu wielokrotnie stosowanego na różnych stronach.
- Umożliwia programistom tworzenie nowych komponentów bez konieczności pisania kodu Javy ani przygotowywania konfiguracji w formacie XML.

Ajax

Framework JSF oferuje wbudowane mechanizmy obsługi elementów technologii Ajax na stronach internetowych. Dzięki temu istnieje możliwość **częściowego przetwarzania widoków** — w takim przypadku

tylko niektóre komponenty widoku są używane do wygenerowania odpowiedzi. Żądania Ajax umożliwiają także **częściowe wyświetlanie strony** (wyświetlanie wybranych komponentów zamiast całej strony).

Obsługę technologii Ajax można włączyć na dwa sposoby:

- programowo (przy użyciu zasobów języka JavaScript);
- deklaratywnie (za pomocą elementu `f:ajax`).

Programowa integracja z technologią Ajax jest możliwa dzięki mechanizmowi obsługi zasobów. Plik *jsf.js* jest predefiniowanym zasobem w ramach biblioteki *javax.faces*. Wspomniany zasób zawiera interfejs API JavaScriptu, który znacznie upraszcza interakcję stron JSF z technologią Ajax. Elementy tego interfejsu można udostępnić w kodzie stron za pomocą znacznika `outputScript`:

```
<h:body>
<!-- ... -->
<h:outputScript
  name="jsf.js"
  library="javax.faces"
  target="body"/>
<!-- ... -->
</h:body>
```

Do wysyłania na serwer asynchronicznych żądań można użyć kodu w tej formie:

```
<h:form prependId="false">
  <h:inputText value="#{user.name}" id="name"/>
  <h:inputText value="#{user.password}" id="password"/>
  <h:commandButton value="Login"
    type="button"
    actionListener="#{user.login}"
    onclick="jsf.ajax.request(this, event, {execute:
      'name password', render: 'status'}); return false;"/>

  <h:outputText value="#{user.status}" id="status"/>
</h:form>
```

W powyższym kodzie:

- Dwa pierwsze pola tekstowe służą do wpisywania nazwy użytkownika i hasła; trzecie pole odpowiada za wyświetlanie statusu (w tym przypadku informacji, czy użytkownik jest zalogowany).
- Atrybutowi `prependId` znacznika formularza przypisano wartość `false`, aby zagwarantować, że identyfikatory poszczególnych elementów zostaną zachowane w oryginalnej formie. W przeciwnym razie framework JSF poprzedziłby identyfikatory elementów potomnych identyfikatorem samego formularza.

- Przycisk polecenia zawiera atrybut `actionListener` identyfikujący metodę klasy komponentu, która ma być wywoływana w odpowiedzi na zdarzenia kliknięcia tego przycisku. Zamiast oczekiwać typowej odpowiedzi powodującej wygenerowanie i wyświetlenie innej strony, funkcja `jsf.ajax.request` powoduje wysłanie asynchronicznego żądania na serwer. Żądanie jest tworzone w odpowiedzi na zdarzenie `onclick` (kliknięcia) przycisku polecenia. Po słowach `execute` i `render` przekazano identyfikatory komponentów oddzielone spacjami. Słowo `execute` poprzedza listę komponentów wejściowych, dla których zostaną wywołane metody ustawiające klasy komponentu; po słowie `render` przekazano listę komponentów, które należy wyświetlić po otrzymaniu asynchronicznej odpowiedzi.

Możliwość przetwarzania tylko części widoku (w tym przypadku elementów `name` i `password`) określa się mianem częściowego przetwarzania widoku (ang. *partial view processing*). Podobnie, wyświetlanie tylko części strony wynikowej (w tym przypadku tylko elementu `status`) określa się mianem częściowego wyświetlania wyniku (ang. *partial output rendering*).

Obsługiwane wartości atrybutu `render` opisano w tabeli 7.3.

Tabela 7.3. Wartości atrybutu `render` znacznika `f:ajax`

Wartość	Opis
@all	Wszystkie komponenty na danej stronie
@none	Żaden z komponentów na stronie (@none jest wartością domyślną)
@this	Element, który spowodował dane żądanie
@form	Wszystkie komponenty należące do danego formularza
Identyfikatory	Identyfikatory komponentów oddzielone spacjami
Wyrażenie EL	Wyrażenie języka EL, które po przetworzeniu reprezentuje kolekcję łańcuchów

Atrybutowi `execute` można przypisać podobny zbiór wartości, jednak w jego przypadku wartością domyślną jest `@this`.

- Komponent `User` zawiera pola, metody ustawiające i zwracające oraz prostą metodę biznesową:

```
@Named
@SessionScoped
public class User implements Serializable {
    private String name;
    private String password;
```

```

private String status;

. . .

public void login(ActionEvent evt) {
    if (name.equals(password))
        status = "Prawidłowe logowanie";
    else
        status = "Nieudane logowanie";
}
}

```

Warto zwrócić uwagę na sygnaturę metody login. Metoda musi zwracać typ void i otrzymywać na wejściu (za pośrednictwem jedynego parametru) obiekt klasy javax.faces.event.ActionEvent.

Deklaratywna integracja z technologią Ajax jest możliwa dzięki elementowi f:ajax. Znacznik f:ajax można albo umieścić wewnątrz komponentu (dzięki temu asynchroniczne żądania Ajax będą stosowane tylko dla tego komponentu), albo wykorzystać do opakowania wielu komponentów (wówczas żądania Ajax będą stosowane dla tych komponentów).

Aby zastosować ten styl integracji z technologią Ajax, poprzedni przykład kodu należałoby przebudować w ten sposób:

```

<h:form prependId="false">
    <h:inputText value="#{user.name}"
        id="name"/>
    <h:inputText value="#{user.password}"
        id="password"/>
    <h:commandButton value="Login"
        type="button"
        actionListener="#{user.login}">
        <f:ajax execute="name password"
            render="status"/>
    </h:commandButton>

    <h:outputText value="#{user.status}"
        id="status"/>
</h:form>

```

W tym kodzie użyto znacznika f:ajax do zdefiniowania listy elementów wejściowych (za pomocą atrybutu execute) oraz elementów wyjściowych do wyświetlenia (za pomocą atrybutu render). Jeśli znacznik f:ajax jest zagnieżdżony w ramach pojedynczego komponentu i jeśli nie wskazano żadnego zdarzenia, asynchroniczne żądanie jest generowane w odpowiedzi na domyślne zdarzenie komponentu macierzystego (w przypadku przycisku polecenia takim zdarzeniem jest onclick).

Znacznik `f:ajax` można stosować także dla wielu komponentów:

```
<f:ajax listener="#{user.checkFormat}">
  <h:inputText value="#{user.name}" id="name"/>
  <h:inputText value="#{user.password}" id="password"/>
</f:ajax>
```

Tym razem w znaczniku `f:ajax` zdefiniowano atrybut `listener` wskazujący odpowiednią metodę Javy:

```
public void checkFormat(AjaxBehaviorEvent evt) {
    // ...
}
```

Metoda nasłuchująca jest wywoływana dla domyślnego zdarzenia elementów potomnych (w przypadku elementu `h:inputText` będzie to zdarzenie `valueChange`). Dodatkowe mechanizmy na bazie technologii Ajax można wskazać także dla elementów potomnych — wystarczy użyć zagnieżdżonych znaczników `f:ajax`.

Żądanie HTTP GET

Framework JSF oferuje obsługę odwzorowywania parametrów żądania GET protokołu HTTP (zawartych w adresie URL) na wyrażenia języka EL. JSF umożliwia też generowanie adresów URL przystosowanych do żądań GET.

Parametry widoku mogą być używane do odwzorowywania parametrów zawartych w adresie URL (składających się na żądanie GET) na elementy języka wyrażeń EL. W tym celu należy umieścić w kodzie strony faceletów następujący fragment:

```
<f:metadata>
  <f:viewParam name="name" value="#{user.name}"/>
</f:metadata>
```

Skutki dostępu do aplikacji internetowej za pośrednictwem adresu `index.xhtml?name=jack` opisano poniżej:

- Uzyskujemy parametr żądania nazwany `name`.
- W razie konieczności konwertujemy i sprawdzamy poprawność tego parametru. Możemy to zrobić za pomocą elementów `f:converter` i `f:validator` (tak jak w przypadku każdego pola typu `h:inputText`). Przykład takiego rozwiązania pokazano poniżej:

```
<f:metadata>
  <f:viewParam name="name" value="#{user.name}">
    <f:validateLength minimum="1" maximum="5"/>
  </f:viewParam>
</f:metadata>
```

```

    </f:viewParam>
</f:metadata>

```

- Jeśli wartość jest prawidłowa, parametr jest kojarzony z polem `#{user.name}`.

Parametry widoku można przetwarzać także bezpośrednio przed wyświetleniem strony (za pomocą elementu `f:event`):

```

<f:metadata>
  <f:viewParam name="name" value="#{user.name}">
    <f:validateLength minimum="1" maximum="5"/>
  </f:viewParam>
  <f:event type="preRenderView"
    listener="#{user.process}"/>
</f:metadata>

```

W powyższym kodzie metoda identyfikowana przez wyrażenie `#{user.process}` może zostać użyta do ewentualnej inicjalizacji przed właściwym wyświetleniem strony.

Adresy URL przystosowane do obsługi żądań GET są generowane za pomocą elementów `h:link` i `h:button`. Zamiast ręcznego konstruowania adresu URL wskazano odpowiednią stronę facetów:

```

<h:link value="Zaloguj" outcome="login"/>

```

Ten zapis jest tłumaczony na następujący znacznik języka HTML:

```

<a href="../../../faces/login.xhtml">Zaloguj</a>

```

Równie łatwo można zdefiniować parametry widoku:

```

<h:link value="Zaloguj" outcome="login">
  <f:param name="name" value="#{user.name}"/>
</h:link>

```

Jeśli w powyższym kodzie z wyrażeniem `#{user.name}` zostanie powiązana wartość "Jacek", przytoczony fragment zostanie przetłumaczony na następujący znacznik HTML-a:

```

<a href="../../../faces/login.xhtml?name=Jacek">Zaloguj</a>

```

Podobnie, istnieje możliwość określenia wyniku za pomocą elementu `h:button`:

```

<h:button value="login"/>

```

Kod w tej formie wygeneruje następujący znacznik języka HTML:

```

<input
  type="button"
  onclick="window.location.href='/JSFSample/faces/index.xhtml'; return
false;"
  value="login" />

```

Punkty rozszerzeń serwera i klienta

Konwertery, obiekty nasłuchujące i mechanizmy sprawdzania poprawności to obiekty **dołączane po stronie serwera**, których celem jest wprowadzanie dodatkowych funkcji do komponentów umieszczonych na stronie. Zachowania to z kolei punkty rozszerzeń strony klienckiej, które uzupełniają treść wyświetlanego komponentu o dodatkowe skrypty.

Konwerter odpowiada za konwersję danych wpisanych w komponencie z jednego formatu na inny format (na przykład z łańcucha na liczbę). Framework JSF udostępnia wiele wbudowanych konwerterów, w tym `f:convertNumber` i `f:convertDateTime`. Konwertery można łatwo stosować dla wszystkich komponentów z możliwością edycji:

```
<h:form>
    Wiek: <h:inputText value="#{user.age}" id="age">
        <f:convertNumber integerOnly="true"/>
    </h:inputText>
    <h:commandButton value="Wyślij"/>
</h:form>
```

Tekst wpisany w polu tekstowym zostanie przekonwertowany (o ile będzie to możliwe) na liczbę całkowitą. Jeśli konwersja tekstu będzie niemożliwa, zostanie wyświetlony komunikat o błędzie.

Istnieje też możliwość utworzenia niestandardowego konwertera:

```
@FacesConverter("myConverter")
public class MyConverter implements Converter {

    @Override
    public Object getAsObject(
        FacesContext context,
        UIComponent component,
        String value) {
        // ...
    }

    @Override
    public String getAsString(
        FacesContext context,
        UIComponent component,
        Object value) {
        // ...
    }
}
```

W tym przypadku metody `getAsObject` i `getAsString` wykonują odpowiednio konwersje obiektu na łańcuch i łańcucha na obiekt pomiędzy obiektami modelu danych a ich reprezentacjami łańcuchowymi

(potrzebnymi do wyświetlania na stronie). Zastosowany obiekt POJO implementuje interfejs Converter i jest oznaczony adnotacją @FacesConverter. Funkcję konwertera może pełnić także strona JSF:

```
<h:inputText value="#{user.age}" id="age">
    <f:converter converterId="myConverter"/>
</h:inputText>
```

Wartość atrybutu value adnotacji @FacesConverter musi być zgodna z wartością użytego powyżej atrybutu converterId.

Mechanizm sprawdzania poprawności danych (tzw. walidator) służy do weryfikacji danych wpisanych za pośrednictwem komponentów wejściowych. Framework JSF udostępnia wiele wbudowanych mechanizmów sprawdzania poprawności, w tym f:validateLength i f:validateDoubleRange. Mechanizmy sprawdzania poprawności można łatwo stosować dla wszystkich komponentów z możliwością edycji:

```
<h:inputText value="#{user.name}" id="name">
    <f:validateLength min="1" maximum="10"/>
</h:inputText>
```

W powyższym kodzie określono, że długość łańcucha wpisywanego w polu tekstowym musi się mieścić w przedziale od 1 do 10 znaków. Jeśli długość danych nie będzie należała do tego przedziału, zostanie wyświetlony komunikat o błędzie.

Istnieje też możliwość utworzenia niestandardowego mechanizmu sprawdzania poprawności:

```
@FacesValidator("nameValidator")
public class NameValidator implements Validator {

    @Override
    public void validate(
        FacesContext context,
        UIComponent component,
        Object value)
        throws ValidatorException {
        // ...
    }
}
```

W powyższym kodzie metoda validate zwraca sterowanie, pod warunkiem że sprawdzana wartość jest poprawna. W przeciwnym razie metoda zgłasza wyjątek ValidatorException. Mechanizmy sprawdzania poprawności można stosować dla dowolnego komponentu z możliwością edycji:

```
<h:inputText value="#{user.name}" id="name">
    <f:validator id="nameValidator"/>
</h:inputText>
```

Wartość atrybutu `value` adnotacji `@FacesValidator` musi być zgodna z wartością atrybutu `id` elementu `f:validator`.

Framework JSF dodatkowo oferuje wbudowane mechanizmy integracji z ograniczeniami definiowanymi na bazie frameworku Bean Validation. Zadanie programisty sprowadza się do umieszczania odpowiednich adnotacji w kodzie komponentu (definiowanie ograniczeń nie wymaga żadnych dodatkowych czynności). Ewentualne komunikaty o błędach (wskutek naruszenia ograniczeń) są automatycznie konwertowane na obiekty klasy `FacesMessage` i prezentowane użytkownikowi końcowemu. Do definiowania grup sprawdzania poprawności można użyć atrybutu `validationGroups` znacznika `f:validateBean` — w ten sposób można wskazać dodatkowe komponenty do sprawdzenia podczas weryfikacji określonego komponentu. Wspomniane rozwiązania zostaną szczegółowo wyjaśnione w rozdziale poświęconym specyfikacji Bean Validation.

Obiekt nasłuchujący nasłuchuje zdarzeń dotyczących komponentu. Zdarzenie może polegać na zmianie wartości, kliknięciu przycisku, kliknięciu linku lub dowolnym innym działaniu. Obiekt nasłuchujący może mieć postać metody komponentu zarządzanego lub samej klasy.

Obiekt `ValueChangeListener` można zarejestrować dla dowolnego komponentu z możliwością edycji:

```
<h:inputText value="#{user.age}"
             id="age"
             valueChangeListener="#{user.nameUpdated}">
```

W powyższym kodzie metoda `nameUpdated` komponentu `User` jest wywoływana w momencie wysyłania powiązanego formularza. Obiekt nasłuchujący na poziomie klasy można utworzyć, implementując interfejs `ValueChangeListener`. Odpowiedni obiekt można wskazać w kodzie strony za pomocą znacznika `f:valueChangeListener`.

W przeciwieństwie do konwerterów, mechanizmów sprawdzania poprawności i obiektów nasłuchujących zachowanie rozszerza mechanizmy strony klienckiej, ponieważ umożliwia deklaratywne dołączanie skryptów. Na przykład znacznik `f:ajax` zdefiniowano jako zachowanie strony klienckiej. Opisany mechanizm dodatkowo umożliwia weryfikację poprawności danych po stronie klienta, rejestrowanie zdarzeń w dzienniku po stronie klienta itp.

Niestandardowe zachowanie można definiować przez rozszerzanie klasy `ClientBehaviorBase` i oznaczanie klas potomnych adnotacją `@FacesBehavior`.

Reguły nawigacji

Framework JSF definiuje niejawne i jawne reguły nawigacji.

Niejawne reguły nawigacji dotyczą wyników pewnych czynności (na przykład kliknięcia linku lub przycisku). Jeśli zostanie znaleziona strona faceletów pasująca do tej akcji, właśnie ta strona zostanie wyświetlona.

```
<h:commandButton action="login" value="Zaloguj"/>
```

W tym przypadku kliknięcie przycisku spowoduje wyświetlenie strony *login.xhtml* znajdującej się w tym samym katalogu.

Jawne reguły nawigacji można definiować za pośrednictwem elementu `<navigation-rule>` w pliku *faces-config.xml*. Do definiowania warunkowych reguł nawigacji służy znacznik `<if>`:

```
<navigation-rule>
  <from-view-id>/index.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/login.xhtml</to-view-id>
    <if>#{user.isPremium}</if>
  </navigation-case>
</navigation-rule>
```

W powyższym kodzie nawigacja pomiędzy stronami *index.xhtml* i *login.xhtml* ma miejsce tylko w sytuacji, gdy użytkownik należy do grupy ważnych klientów.

A

adnotacja, 12

- @Alternative, 86, 95
- @ApplicationPath, 136
- @AroundInvoke, 89
- @AssertFalse, 155
- @AssertTrue, 155
- @Asynchronous, 74
- @Constraint, 159
- @Context, 141
- @CookieParam, 141
- @DecimalMax, 156
- @DecimalMin, 155
- @DELETE, 138
- @DenyAll, 42
- @DependsOn, 67
- @Digits, 156
- @Discriminator, 48
- @EJB, 65
- @ElementCollection, 47
- @Embeddable, 47
- @Embedded, 47
- @Entity, 46
- @FacesBehavior, 117
- @Fancy, 84
- @FormParam, 137, 141
- @Future, 157
- @GroupSequence, 164
- @HEAD, 139
- @HeaderParam, 141
- @HttpConstraint, 41
- @HttpMethodConstraint, 41
- @Inheritance, 48
- @Inject, 83, 86
- @Interceptor, 89
- @InterceptorBinding, 88
- @ManyToMany, 48

- @ManyToOne, 48
- @MapKey, 49
- @MapKeyClass, 49
- @MapKeyColumn, 49
- @MappedSuperclass, 48
- @MatrixParam, 141
- @Max, 156
- @MessageDriven, 68, 152
- @Min, 155
- @MultipartConfig, 44, 45
- @Named, 85, 102
- @NamedQuery, 55
- @NotNull, 155, 159
- @Null, 155
- @OneToMany, 48
- @OneToOne, 48
- @Oneway, 123
- @OPTIONS, 139
- @Past, 157
- @Path, 13, 135
- @PathParam, 135
- @Pattern, 157, 159
- @PermitAll, 42
- @POST, 137
- @PostActivate, 64
- @PrePassivate, 64
- @Provider, 140
- @Qualifier, 86
- @QueryParam, 136
- @Remote, 63, 65
- @Remove, 63
- @RolesAllowed, 42
- @Schedule, 19, 75
- @ServiceMode, 125
- @ServletSecurity, 40
- @Singleton, 12
- @Size, 156, 159
- @SOAPBinding, 123

adnotacja
 @SQLResultSetMapping, 53
 @Startup, 67
 @Stateful, 12, 63
 @Stateless, 12, 65
 @Stereotype, 94
 @Target, 89, 158
 @Timeout, 75
 @TransactionAttribute, 72
 @TransactionManagement, 71
 @TransportProtected, 42
 @Valid, 57, 161
 @WebFault, 123
 @WebFilter, 32
 @WebInitParam, 32
 @WebListener, 33
 @WebService, 121
 @WebServiceProvider, 124
 @WebServlet, 12, 26
 @XmlRootElement, 135
 @ZipCode, 159
 FetchType.EAGER, 48
 FetchType.LAZY, 48

adres URL, 113

Ajax, 109

aktualizowanie encji, 55

aktywacja komponentu, 64

aplikacje korporacyjne, 14

asynchroniczne żądanie, 130

atak XSS, 29

atrybut

 country, 159

 execute, 111

 group, 159

 payload, 159

 render, 111

atrybuty

 @WebService, 121

 @WebServiceProvider, 124

automatyczna weryfikacja, 56

B

Bean Validation, 18, 56, 154

bezpieczeństwo, 40

bezpieczeństwo wątków, 50

bezzastawowy komponent sesyjny, 65, 74

biblioteki znaczników, 101

blokada Write, 67

blokady

 optymistyczne, 59

 pesymistyczne, 59

blokowanie encji, 59

błąd nieprawidłowej zależności, 86

błędy, 44

buforowanie, 60

C

CDI, Contexts and Dependency

 Injection, 14, 82

ciało komunikatu, 132

CRUD, create, read, update, delete, 52

CSS, Cascading Style Sheets, 101

cykl życia

 encji, 52

 obiektów, 33

 serwletu, 28

częściowe

 przetwarzanie widoków, 109

 wyświetlanie strony, 110

D

definiowanie

 komponentu kompozytowego, 106

 ograniczeń, 41

 zasięgów, 94

deklarowanie

 filtrów, 33

 serwletów, 31

dekoratory, 91

deskryptor wdrożenia

 aplikacji internetowej, 28

 web.xml, 29

dezaktywacja komponentu, 64

dołączanie

 arkusza stylów, 105

 skryptu, 105

dostawca JMS, 144, 149

dostęp do

- serwletów, 40

- zasobu, 141

DRY, Don't Repeat Yourself, 54

dzielenie konfiguracji, 18

E

EJB, Enterprise JavaBeans, 13, 62

EJB.Lite, 81

EL, Expression Language, 102

element

- <absolute-ordering>, 39

- <after>, 40

- <alternatives>, 86

- <async-supported>, 37

- <auth-constraint>, 41

- <before>, 40

- <error-page>, 44

- <exception-type>, 44

- <filter>, 32

- <filter-mapping>, 32

- <interceptors>, 90

- <name>, 39

- <navigation-rule>, 118

- <ordering>, 39, 40

- <others/>, 39

- <security-constraint>, 41

- <user-data-constraint>, 41

- <web-resource-collection>, 41

- cc:implementation, 107, 109

- cc:interface, 107, 108

- f:ajax, 111

- f:converter, 113

- f:event, 114

- f:validateBean, 117, 163

- f:validator, 113

- h:button, 114

- h:link, 114

- html, 101

- outputScript, 110

- shared-cache-mode, 60

- validation-mode, 165

elementy WSDL, 127

encje, 46

Enterprise JavaBeans, 19

F

fabryka menedżera encji, 165

facecety, 21, 101

filtr LoggingFilter, 32

filtry serwletów, 32

formularz logowania, 106

fragment komunikacji, web

- fragment, 39

framework

- Bean Validation, 117

- JSF, 100, 166

- JSF2, 106

- OSGi, 18

G

globalne nazwy JNDI, 70

grupa Default, 165

grupy sprawdzania poprawności,
162, 165

I

implementacja

- punktu końcowego, 126

- referencyjna, 15

instancja kontekstowej, 82

interfejs

- API Embeddable EJB, 19

- API EntityTransaction, 58

- AsyncListener, 36

- BindingProvider, 128

- Criteria API, 53

- Dispatch, 129

- Embeddable API, 79

- Endpoint, 125

- Event, 96

- Greeting, 83, 84

- HttpServletRequest, 42

- HttpSession, 30

- HttpSessionActivationListener, 35

- HttpSessionAttributeListener, 35

- HttpSessionBindingListener, 35

- HttpSessionListener, 34

- JPA, 46

interfejs

- JTA, 58
- LogicalHandler, 131
- MessageBodyReader, 140
- MessageBodyWriter, 140
- MessageListener, 68, 152
- Provider, 124
- Provider<DataSource>, 124
- Provider<SOAP message>, 125
- Provider<Source>, 125
- Servlet, 27
- ServletContextAttributeListener, 34
- ServletRequestAttributeListener, 36
- ServletRequestListener, 36
- SPI, 97
- TimedObject, 77

J

- Java EE, 12
- JavaServer Faces, 100
- JAXB, Java API for XML Binding, 120
- JAXR, Java API for XML Registries, 13
- JAX-RPC, Java API for XML-based RPC, 13
- JAX-RS, Java API for RESTful Web Services, 13, 20, 134
- JAX-WS, Java API for XML Web Services, 13, 119
- JCP, Java Community Process, 14
- jednostka utrwalania, 50
- język
 - deklarowania widoków, 101
 - WSDL, 119
 - wyrażeń EL, 102
 - XHTML 1.0, 101
 - zapytań JPQL, 53
 - zapytań SQL, 53
- JMS, Java Message Service, 68, 144
- JPA, Java Persistence API, 21, 46, 164
- JPQL, Java Persistence Query Language, 53
- JSF, JavaServer Faces, 13, 21, 100, 166
- JSR, Java Specification Request, 15

K

- katalog
 - META-INF, 39
 - resources, 43
- klasa
 - Application, 136
 - FacesMessage, 117
 - GreetingService, 84
 - MessageContext, 132
 - Order, 135
 - Response, 143
 - Service, 127
- klasy
 - dostępne do wbudowania, 47
 - mechanizmu odwzorowującego, 143
- klient
 - JMS, 144
 - usługi sieciowej, 127
- klucz
 - do platformy, 168
 - główny, primary key, 46
 - obcy, foreign key, 46
- komponent
 - Foo, 67
 - ServletFilter, 14
 - ServletListener, 14
 - User, 111
- komponenty, 13
 - dostępne lokalnie, 63, 65
 - EJB, 19
 - encyjne, 62
 - sesyjne, 62
 - sterowane komunikatami, 68
 - kompozytowe, 106
 - MDB, 68
 - sesyjne
 - bezstanowe, 65
 - singletonowe, 66
 - stanowe, 62
 - zarządzane, Managed Beans, 19, 23
 - złożone, 105
- komunikat, 123, 132, 144
 - JMS, 144
 - ciało, 146
 - nagłówek, 145
 - właściwości, 145
 - o błędzie, 117, 159

- konfiguracja encji, 51
- konfigurowanie kontenera EJB, 80
- kontekst, 14, 22, 82, 92
 - cyklu życia, 82
 - serwletów, 29
 - utrwalania, 50, 52
- kontener
 - EJB, 80
 - serwletów, 26
- konwerter, 115
 - f:convertDateTime, 115
 - f:convertNumber, 115
- konwertery niestandardowe, 115
- koperta, envelope, 119
- kwalifikator, 84
- kwalifikatory CDI, 85

L

- liczniki czasu, 75

Ł

- łańcuch obiektów, 133

M

- MDB, message-driven bean, 68
- mechanizm
 - adnotacji, 12
 - sprawdzania poprawności, 116
 - rozszerzeń, 14
- menedżer encji, 50, 51
- metadane, 95
- metoda
 - addFilter, 33
 - addServlet, 31
 - AsyncContext.complete, 37
 - authenticate, 42
 - cleanupResources, 25
 - consumer.receive, 151
 - doGet, 38
 - ejbTimeout, 77
 - EntityManager.merge, 55
 - EntityManager.remove, 56
 - EntityManager.rollback, 59

- getCustomerNames, 103
- getOrder, 136, 143
- getParts, 45
- getPort, 128
- handleFault, 132
- handleMessage, 132
- HttpServletRequest.getRequestDispatcher, 30
- HttpServletResponse.sendRedirect, 30
- init, 28
- login, 42
- onMessage, 68, 152
- Persistence.createEntityManager
 - ↳ Factory, 165
- PostConstruct, 66
- PreDestroy, 66
- producenta, 86
- putXml, 138
- ServletContainerInitializer
 - ↳ onStartUp, 31, 36
- ServletContext.getContext, 30
- ServletContext.getRequestDispatcher, 30
- ServletContextListener
 - ↳ contextInitialized, 36
- ServletRegistration.setAsyncSupported, 37
- session.getAttribute, 30
- session.setAttribute, 30
- setupResources, 25
- validate, 116
- Validator.validate, 56
- metody
 - biznesowe, 123
 - protokołu HTTP, 137
 - zasobu, 140
- model
 - najpierw kod, code first, 121
 - najpierw kontrakt, contract first, 121
 - publikacja-subskrypcja, 145
 - punkt-punkt, 144
- MOM, Message-oriented middleware, 144
- MVC, Model-View-Controller, 100

N

- nagłówek
 - Accept, 140
 - Content-Type, 140
- narzędzie wscompile, 119
- nasłuchiwanie
 - zdarzeń, 34–36
 - zmian atrybutów, 34–36
- nawiasy klamrowe, 135
- nazwa
 - serwletu, 26
 - tabeli, 46
- nazwy JNDI, 71
- NetBeans, 168

O

- obiekt
 - Application, 142
 - AsyncListener, 38
 - BeanManager, 99
 - Connection, 147
 - EntityManagerFactory, 60
 - HttpHeaders, 142
 - InvocationContext, 89
 - jaxbObject, 132
 - MessageDrivenContext, 70
 - messageProducer, 149
 - Providers, 142
 - QueueBrowser, 151
 - Request, 142
 - RequestDispatcher, 30
 - Response, 130
 - SecurityContext, 142
 - ServletContext, 29
 - session, 148
 - TemporaryQueue, 153
 - UriInfo, 142
 - UserTransaction, 90
 - ValueChangeListener, 117
- obiekty
 - dodłączane po stronie serwera, 115
 - nasłuchujące, 33, 36
 - obsługujące logiczne, 131, 133
 - obsługujące protokołu, 131
 - obsługujące SOAP, 133

- POJO, 12
- przechwytyjące, 21, 88, 90
- tymczasowe miejsc docelowych, 153
- obserwator, observer, 95
- obserwatory transakcyjne, 97
- obsługa
 - buforowania, 60
 - metod protokołu HTTP, 20
 - negocjacji rodzaju treści, 20
 - operacji asynchronicznych, 37
 - powiązania XML, 120
 - technologii Ajax, 21, 110, 112
 - zasięgów, 23
 - zasobów, 105, 110
 - zdarzeń cyklu życia, 23
 - żądań GET, 114
 - żądań protokołu HTTP, 27
 - żądań wieloczęściowych, 44
- odbieranie komunikatu
 - asynchroniczne, 152
 - synchroniczne, 150
- odwołanie do zasobu, 105
- odwzorowania WSDL-Java, 127
- odwzorowywanie
 - danych, 120
 - niestandardowej reprezentacji, 140
 - typów języka, 122
 - wyjątków, 143
- ograniczenia
 - bezpieczeństwa, 41
 - niestandardowe, 22, 158
 - wbudowane, 154
- oprogramowanie pośredniczące
 - MOM, 144
- otrzymywanie komunikatów, 148

P

- pakiet javax.validation.constraints, 154
- pakiet zgodności technologicznej, 15
- pakowanie zasobów, 43
- pamięć podręczna, 61
- platforma Java EE 6, 168
- plik
 - beans.xml, 86, 90
 - ejb-jar.xml, 70
 - faces-config.xml, 21, 118

- JAR, 39
- JSESSIONID, 29
- jsf.js, 110
- library.jar, 43
- login.xhtml, 107
- persistence.xml, 51, 57, 165
- validation.xml, 22
- web.xml, 28
- web-fragment.xml, 20, 39, 40
- pliki
 - .war, 29, 70
 - .xhtml, 105
 - cookie, 29
- podmiot zabezpieczeń, security
 - principal, 75
- POJO, Plain Old Java Object, 12
- poła nagłówka, 146
- powtarzalne adresy URL, 21
- producenci zdarzeń, 95
- profile, 13
- protokół
 - SOAP, 20, 119
 - zatwierdzania dwufazowego, 72
- przenośne rozszerzenie, portable
 - extensions, 97
- przepisywanie adresów URL, 30
- przetwarzanie
 - asynchroniczne, 37, 74
 - współbieżne, 67
- przycinanie, pruning, 13
- punkt wstrzykiwania delegacji, 91
- punkty
 - końcowe, 121, 125
 - końcowe dynamiczne, 124
 - rozszerzeń, 115
 - wstrzykiwania, 83, 85

R

- reguły
 - bezpieczeństwa, 42
 - nawigacji, 118
 - nawigacji warunkowe, 118
 - odwzorowań, 120
- relacje pomiędzy encjami, 48
- REST, 134, 140
- RI, Reference Implementation, 15

- rola manager, 41
- rozszerzenie, 98
- rozszerzony kontekst utrwalania, 52

S

- SEI, Service Endpoint Interface, 121
- serwer
 - GlassFish, 168
 - GlassFish Community, 18
- serwlet
 - AccountServlet, 28
 - FileUploadServlet, 45
- serwlety, 14, 20, 26
- serwlety asynchroniczne, 20
- singletonowy komponent sesyjny, 66
- specyfikacja, 15
 - Bean Validation, 56
 - CDI, 82, 88
 - EJB 3.1, 81
 - EJB.Lite, 81
 - JAX-WS, 120
 - JSR 224, 119
 - JSR 303, 154
 - JSR 311, 134
 - JSR 314, 100
 - JSR 315, 26
 - JSR 316, 15
 - JSR 317, 46
 - JSR 318, 62
 - JSR 914, 144
 - WS-I Attachments Profile, 120
 - WS-I Basic Profile, 120
 - WS-I Simple SOAP Binding
 - Profile, 120
 - WS-Reliable Messaging, 120
 - WS-Secure Conversation, 120
 - WS-Security, 120
- specyfikacje
 - serwletów, 14
 - technologii internetowych, 170
 - technologii korporacyjnych, 170
- SPI, Service Provider Interface, 97
- sprawdzanie poprawności, 116, 162–166
 - encji, 58
 - komponentów, 22

stanowy komponent sesyjny, 62
stereotyp @Decorator, 91
stereotypy, 94
stos technologii, 17
stosowanie strony, 106
strona szablonu, 104
struktura typu Map, 49

Ś

ścieżka do zasobów, 43
śledzenie sesji, 30
środowisko
 IDE NetBeans, 168
 JAX-WS, 119

T

TCK, Technology Compliance Kit, 15
technologia
 Ajax, 110
 Bean Validation, 154
 CDI, 97
 JavaServer Faces, 100
 JAXB, 120
 JAX-RS, 13, 20, 134
 JAX-WS, 13, 119
 JMS, 68, 144
technologie
 internetowe, 15, 170
 korporacyjne, 16, 170
 usług sieciowych, 16, 171
 zarządzania i zabezpieczeń, 17, 171
transakcje, 58, 71
 zarządzane przez komponent, 71
 zarządzane przez kontener, 71
tryb
 dostarczania komunikatów, 149
 potwierdzania otrzymywania
 komunikatów, 148
tworzenie
 liczników czasu, 78
 stereotypów, 95
typ void, 74, 123
typy, 48
typy MIME, 140

U

UEL, Unified Expression Language, 82
unikanie powtórzeń, 54
usługa Timer Service, 77
usługi sieciowe, 18, 119, 171
usługi sieciowe RESTful, 134
usuwanie encji, 56
uwierzytelnianie, 42

W

walidator, 116
warstwa
 prezentacji, 22
 transakcyjna, 22
wartości
 adnotacji @TransactionAttribute,
 73
 validation-mode, 165
wiązanie
 metod, 137
 obiektów z sesją, 30
 żądania z zasobem, 141
właściwości
 ActivationConfig, 69
 BindingProvider, 128
właściwość
 javax.persiste
 nce.cache.storeMode, 61
 javax.persistence.cache.
 ↪ retrieveMode, 61
WSDL, Web Services Description
 Language, 119
wstrzykiwanie
 komponentu, 24, 102
 obiektu, 87
 zależności, 14, 22, 82
wyjątek, 143
 ConstraintViolationException, 57
 org.example.MyException, 44
 SOAPFaultException, 123
 ValidatorException, 116
wyrażenia JPQL, 54
wysyłanie wiadomości, 147
wywołania zwrotne cyklu życia, 25

wzorzec

architektury, 94

fabryki, 87

X

XSS, cross-site scripting, 29

Z

zabezpieczenia serwletów, 20

zagadnienia przecinające, 88

zalety komponentów złożonych, 109

zapytania dynamiczne, 54

zarządzanie instancjami encji, 50

zasada jednego rozwiązania, 14

zasięg

@ApplicationScoped, 93

@ConversationScoped, 93

@Dependent, 93

@RequestScoped, 93

@SessionScoped, 93

zasięgi CDI, 93

zbiór definicji ograniczeń, 22

zdarzenie, 95

AfterBeanDiscovery, 98

AfterDeploymentValidation, 98

BeforeBeanDiscovery, 98

BeforeShutdown, 98

ProcessAnnotatedType, 98

ProcessInjectionTarget, 98

ProcessProducer, 98

znacznik, *Patrz* element

znaczniki faceletów, 101, 103

Ż

żądania

blokujące, 130

multipart/form-data, 44

protokołu HTTP, 27

żądanie

DELETE, 138

GET, 27, 113

HEAD, 138, 139

OPTIONS, 139

POST, 28

PUT, 138

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Java EE 6. Leksykon kieszonkowy



Korporacyjna wersja Javy (JEE, od ang. Java Enterprise Edition) święci triumfy. Najświeższa odsłona tego języka, oznaczona numerem 6, rozwiązuje problemy znane z poprzednich wersji oraz wprowadza wiele nowości i ulepszeń. Dzięki tym zmianom praca programistów stała się tak przyjemna i wydajna, jak nigdy przedtem. Przejrzyste API, wstrzykiwanie zależności (CDI) oraz dobre wsparcie dla usług sieciowych i formatu REST (JAX-RS) to tylko niektóre z atutów JEE 6.

Ta wyjątkowa książeczka zawiera najważniejsze informacje o JEE 6 – taki wygodny leksykon zawsze możesz mieć pod ręką. W trakcie lektury dowiesz się, jak nowe elementy platformy wpływają na znane wzorce projektowe i jak korzystać z Java Persistence API (JPA). Poznasz także charakterystykę każdego rodzaju ziarna Enterprise JavaBeans (EJB). Ponadto nauczysz się tworzyć usługi sieciowe na podstawie protokołu SOAP i REST oraz korzystać z usług rozsyłających wiadomości (JMS). Na koniec zobaczysz, jak zapewnić integralność Twoim danym z wykorzystaniem BeanValidation (JSR-303).

To obowiązkowa pozycja dla każdego programisty JEE 6. Jeszcze nigdy tak szeroki zbiór informacji nie był dostępny w równie poręcznej formie. Musisz go mieć!

Sprawdź:

- co nowego kryje platforma JEE 6
- jak wstrzykiwać zależności
- do czego służą profile
- jak dostosować aplikację do platformy JEE 6

Wykorzystaj potencjał korporacyjnej wersji języka Java!

helion.pl
księgarnia
internetowa

Nr katalogowy: 13353



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

🔗 <http://helion.pl/promocje>

📖 Książki najchętniej czytane:

🔗 <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

🔗 <http://helion.pl/nawosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po **WIECEJ**



KOD KORZYŚCI

ISBN 978-83-246-6640-9



Cena 29,90 zł