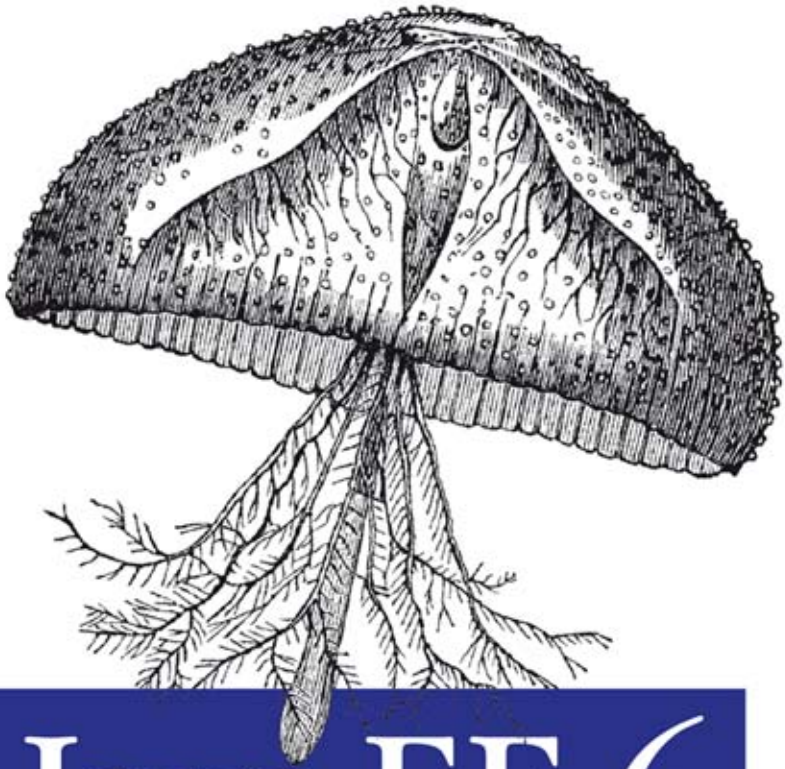
Podręczny przewodnik po JEE 6!



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

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/jee6lk
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

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 wyrażeń 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
`ui:composition`	Definiuje układ strony, który może obejmować szablon. W przypadku zastosowania atrybutu `template` 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.
`ui:insert`	Znacznik stosowany na stronach szablonowych służy do definiowania obszaru wstawiania właściwej treści w szablonie. Odpowiedni znacznik `ui:define` w kodzie strony klienckiej szablonu umożliwia zastąpienie treści.
`ui:define`	Znacznik używany w kodzie strony klienckiej szablonu. Definiuje treść wstawianą w miejscu wyznaczonym przez szablon (za pomocą odpowiedniego znacznika `ui:insert`).
`ui:component`	Umieszcza w drzewie komponentów JSF nowy komponent interfejsu użytkownika. Komponenty i fragmenty treści spoza tego znacznika są ignorowane.
`ui:fragment`	Znaczenie tego znacznika jest podobne jak w przypadku `ui:component`, tyle że treść spoza tego znacznika nie jest ignorowana.
`ui:include`	Dołącza dokument wskazywany przez atrybut `src` (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>Facelety 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: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 konkatenację 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>
```

- 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 faceletów:

```
<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:validate`
`➥DoubleRange`. 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.

Skorowidz

A

B

C

D

E

F

G

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

X

Z

Ż

Programowanie

O'REILLY®

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/nowosci

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

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-246-6640-9

Cena 29,90 zł

Informatyka w najlepszym wydaniu

9788324666409