JSF Core Tags

Jsf_core udostępnia podstawowe (i niezależne od sposobu wyświetlania) znaczniki. Są wśród nich np. tagi konwerterów, walidatorów, listenerów, które można zagnieździć w tagach komponentów, rejestrując na nich konwertery, walidatory itd.

Proste przykłady zarejestrowania konwertera daty, walidatora liczb (by były z przedziału od-do) i listenera:

```
<h:inputText id="userNo"   value="#{UserNumberBean.userNumber}">
        <f:validateLongRange  minimum="#{UserNumberBean.minimum}"
        maximum="#{UserNumberBean.maximum}" />
</h:inputText>

<h:outputText value="#{cashier.shipDate}">
  <f:convertDateTime dateStyle="full" />
</h:outputText>

<h:commandLink id="Germany" action="bookstore"
  actionListener="#{localeBean.chooseLocaleFromLink}">
  <h:outputText value="#{bundle.German}" />
</h:commandLink>
```

Do ciekawszych tagów należą poza tym:

1. **view**

Obowiązkowy! Strona JSF jest reprezentowana jako drzewo, jej korzeniem jest komponent UIViewRoot. Tag view reprezentuje ten komponent, wszystkie pozostałe tagi muszą być w nim zawarte:

```
<f:view>
... inne tagi ...
</f:view>
```

2. **subview**

By załączyć stronę zawierającą tagi JSF należy zagnieździć ją w tagu subview (subview stanie się korzeniem poddrzewa dla znaczników z zaincludowanej strony)

```
<f:subview id="nestedPage">
  <jsp:include page="nestedPage.jsp">
</f:subview>
```

3. **facet**
   Zagnieżdżony komponent o specjalnym znaczeniu dla znacznika-rodzica - jak nagłówek, stopka itp. Ten tag może mieć tylko jedno dziecko, więc jeśli chcielibyśmy zagnieździć więcej tagów, musimy w nim zagnieździć panelGroup i dopiero w nim pozostałe.

```
<h:column >
  <f:facet name="header">
    <h:outputText value="#{bundle.ItemQuantity}" />
  </f:facet>
  <h:inputText id="quantity" size="4"
    value="#{item.quantity}" >
  </h:inputText>
</h:column>
```

4. **loadBundle** - do lokalizacji aplikacji - pozwala określić, gdzie jest ResourceBundle, z którego będziemy chcieli korzystać

```
<f:loadBundle var="bundle" basename="carstore.bundles.Resources"/>
```

by potem użyć zlokalizowanych wiadomości w zagnieżdżonych tagach:

```
<h:outputText value="#{bundle.Hello}"/>
selectItem, selectItems - zbiór elementów i jeden element ze zbioru:
<h:selectOneMenu   id="shippingOption"   required="true"
                          value="#{cashier.shippingOption}">
        <f:selectItem  itemValue="2"  itemLabel="#{bundle.QuickShip}"/>
         <f:selectItem   itemValue="5"  itemLabel="#{bundle.NormalShip}"/>
</h:selectOneMenu>

<h:selectManyCheckbox   id="newsletters"   layout="pageDirection"
 value="#{cashier.newsletters}">
        <f:selectItems  value="#{newsletters}"/>
</h:selectManyCheckbox>
```

JSF HTML Tags

Biblioteka jsf_html udostępnia HTML-owe elementy GUI, znaczniki dla wszystkich kombinacji UIComponent i rendererów z HTML RenderKit zdefiniowanych w specyfikacji JavaServer Faces. Jednemu komponentowi może odpowiadać więcej niż jeden znacznik, np. zarówno commandButton, jak commandLink odpowiadają funkcjonalności UICommand.

Ważniejsze znaczniki:

1. **commandButton, commandLink (UICommand):**

komponent wykonujący akcję. Można dla niego określić akcję (wprost lub jako metodę beana)

```
<h:commandButton id="submit"   action="#{userNumberBean.getOrderStatus}" value="Submit" />
<h:commandButton value="#{bundle.Submit}"   action="success"/>
```

2. **dataTable, column (UIData)**

dataTable służy do iteracji po danych: liście/tablicy beanów bądź pojedynczym beanie, java.sql.ResultSet/RowSet etc. (generalnie: czymś, co ma javax.faces.model.DataModel); column reprezentuje kolumnę

```
<h:dataTable id="items"   value="#{cart.items}"   var="item">
  <h:column >
       <h:inputText id="quantity" size="4"  value="#{item.quantity}" >       </h:inputText>
   </h:column>
</h:dataTable>
```

value mówi, po czym się iterujemy, var pozwala nazwać ,,bieżący'' element, by odwoływać się do niego; można ustawić, które rzędy chcemy wyświetlać przez first (numer pierwszego wyświetlanego) i rows (liczba wyświetlanych)

3. **message, messages (UIMessage, UIMessages)**

Informacje o błędach - message wyświetli błąd dla pojedynczego komponentu (określonego przez atrybut for), messages wszystkie błędy dla strony.

```
<h:message   style= "color: red;   font-family: 'New Century Schoolbook', serif;
 font-style: oblique;   text-decoration: overline" id="errors1" for="userNo"/>
<h:inputText id="userNo" value="#{UserNumberBean.userNumber}">
 <f:validateLongRange minimum="0" maximum="10" />
```

4. **panelGrid**

   wyświetla tabelkę, panelGroup - zbiera komponenty pod jednym rodzicem (np. tworząc rząd tabelki)

```
<h:panelGrid columns="2"
 styleClass="list-background"
 title="#{bundle.Checkout}">
 <h:outputText value="#{bundle.Name}" />
 <h:inputText id="name" size="50"
   value="#{cashier.name}"
   required="true">
   <f:valueChangeListener
     type="listeners.NameChanged" />
 </h:inputText>
 <h:message styleClass="validationMessage" for="name"/>
 <h:outputText value="#{bundle.CCNumber}"/>
</h:panelGrid>
```

Pod adresem http://www.horstmann.com/corejsf/jsf-tags.html są opisy tagów - z listami atrybutów do poszczególnych znaczników i przykładami użycia, wygenerowanego dla tagów html-a i sposobu ich wyświetlania,

Pewna liczba własności jest wspólna dla wszystkich (czy prawie wszystkich) komponentów, są to np.:

id - unikalna nazwa komponentu; jeśli nie będzie podana, a będzie potrzebna (czyli inny komponent/klasa po stronie serwera odwołują się do tego komponentu) implementacja JSF wygeneruje ją automatycznie

value - wartość komponentu

style - styl CSS odnoszący się do komponentu, może być wklepany nawet bezpośrednio:

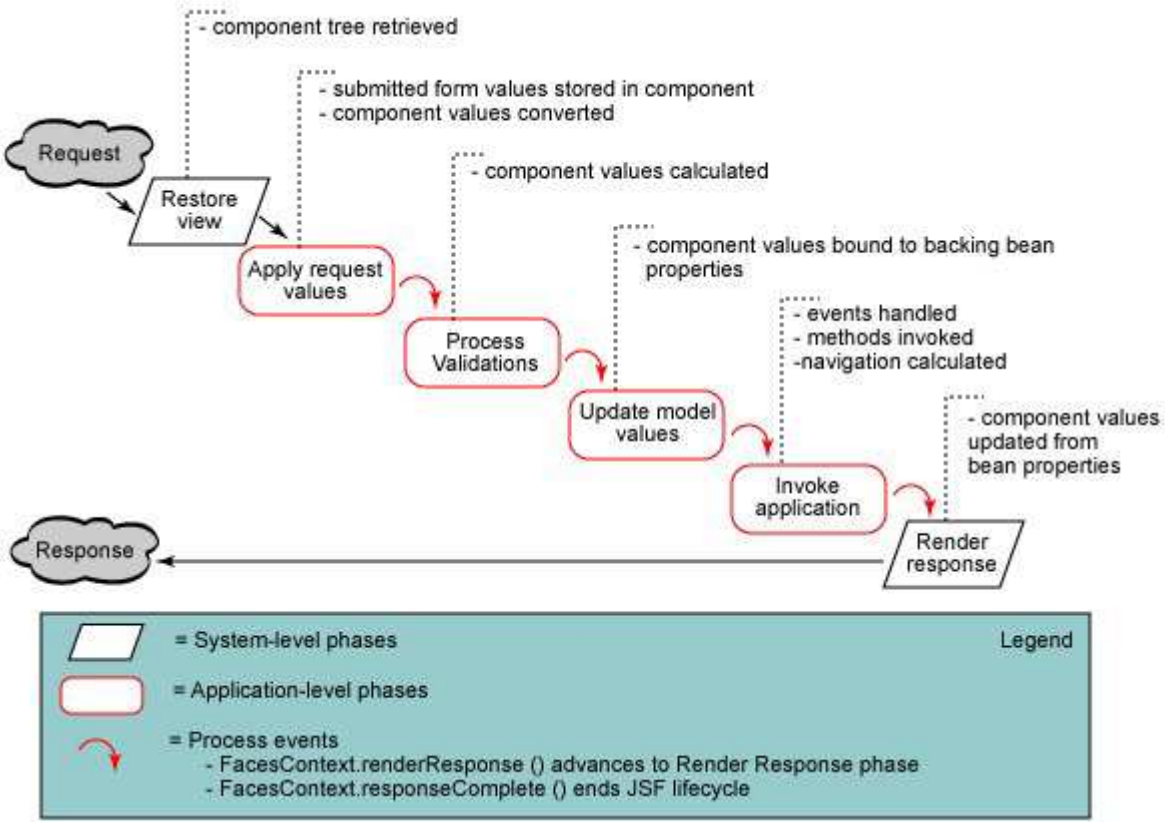
```
<h:outputText style= "color: red" value="#{UserNumberBean.minimum}"/>
```

styleClass - klasa CSS zawierająca definicję stylu:

```
<h:outputText styleClass="subtitle"  value="just a subtitle"/>
```

immediate - czy wyzwalane zdarzenia, walidacja czy konwersja (zależnie od komponentu) mają następować podczas Apply Request Value Phase czy później, w zwykłej (patrz rysunek poniżej) kolejności.

Np. ustawienie immediate na true zarówno dla linka, jak i dla jakiegoś pola, którego wartość można wprowadzić spowoduje, że nowa wartość pola będzie dostępna dla zdarzeń odpalonych przez kliknięcie na link (po wprowadzeniu nowej wartości). W podobnym scenariuszu jeśli dla pola nie ustawimy immediate (false jest defaultem), dla zdarzeń odpalonych przez link dostępna będzie stara wartość pola.



binding - związuje komponent z property beana:

```
<inputText binding="\#{UserNumberBean.userNoComponent}" />
```

converter, validator – jak nazwa wskazuje

```
<h:inputText id="userNo"   value="\#{UserNumberBean.userNumber}"
```

```
        validator="\#{UserNumberBean.validate}" />
```

rendered - definiuje warunki, przy których komponent ma być wyświetlany

```
            <h:commandLink id="check" rendered="#{cart.numberOfItems > 0}">
                <h:outputText value="#{bundle.CartCheck}"/>
             </h:commandLink>
```