Kennismaking met JSF

Heeft Java eindelijk zijn Oracle Forms?

In een serie van drie artikelen zal Lucas Jellema een rondleiding geven door de wereld van Java Server Faces. In dit eerste deel kijken we naar de basis-elementen binnen de JSF standaard. Als IDE maken we gebruik van Oracle JDeveloper 10.1.3.1, hoewel dit eerste deel geen Oracle-specifieke faciliteiten gebruikt. Hoewel enige Java-achtergrond nuttig is, zijn de artikelen toegankelijk voor alle Oracle-ontwikkelaars.

Het volgende deel gaat in op de Oracle implementatie van JSF: ADF Faces. Hierin kijken we onder meer naar de ingebouwde AJAX-faciliteiten van ADF Faces. Ook bespreken we dan de data-binding faciliteiten van ADF Faces. Het derde en laatste deel gaat in op andere JSF componenten library's zoals Apache MyFaces en IceFaces, het combineren van verschillende library's en de ontwikkeling van eigen JSF componenten.

Feestje

Het ontwikkelen van webapplicaties met Java/J2EE-technologie was de afgelopen jaren niet altijd een feestje. Het vele, complexe werk met vaak teleurstellende resultaten nodigden organisaties en ontwikkelaars gewend aan productieve, rijke grafische ontwikkeltools als Oracle Forms bepaald niet uit. Java Server Faces zou daar verandering in kunnen brengen.
Java Server Faces vormt de volgende stap in een lange ontwikkeling van webapplicatie-technologie, die ons sinds de jaren negentig van CGI langs Servlet en JSP voerde naar JSTL, EL (Expression Language) en Struts. Deze ontwikkeling bracht ons weliswaar steeds meer productiviteit, steeds minder hoge eisen aan de ontwikkelaar, steeds betere ondersteuning van IDE's en steeds rijkere user interfaces, maar schoot nog steeds te kort ten opzichte van traditionele, proprietary tools als Oracle Forms, Delphi en PowerBuilder.
Met JSF is er eindelijk is er een standaard J(2)EE-technologie die wel uitnodigend is: productief, gebaseerd op componenten, een declaratieve, visuele, 4GL-vorm van ontwikkelen (property palette, drag & drop, WYSIWYG editor), een mechanisme van events en triggers en programmatische toegang tot de pagina-elementen. JSF is onderdeel van het JEE-platform en wordt ondermeer ondersteund door IBM, BEA, Sun, Borland, Oracle en diverse Open Source groepen. Een JSF-applicatie kan op alle serieuze JEE applicatieservers (WebSphere, WebLogic, Oracle AS, JBoss) en servlet-containers (Tomcat) worden uitgerold.
In dit artikel maken we kennis met de kern van JSF door stapje voor stapje de belangrijkste elementen door te nemen. Dit zijn achtereenvolgens UI Componenten, het Model met Managed Beans en Binding attributen en methodes, de Events-en-Listeners infrastructuur, Converters en Validators en Navigation. Aan het eind van het artikel vind je referenties naar aanvullende bronnen, zoals enkele waardevolle JSF-boeken en natuurlijk de broncode voor dit artikel.

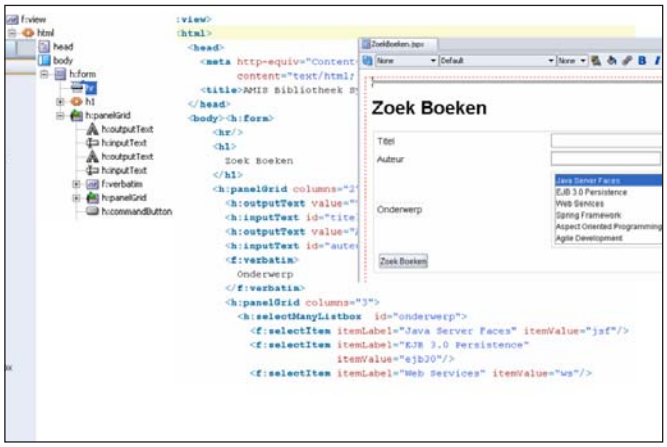
UI-componenten

Aan de basis van JSF staan de UI-componenten. Deze worden veelal gerepresenteerd door Custom JSP Tags in JSP pagina's, als JSF tenminste wordt gebruikt om een HTML-client te bedienen: JSF UI-componenten kunnen speciale renderers hebben – display markup uitspugers – voor allerlei View technologieën zoals XUL, HTC, WML, Telnet, Flash, SVG. In dit artikel houden we het bij HTML.
De JSF-standaard bevat een tamelijk sobere collectie UI Componenten, waaronder HtmlForm, HtmlInputText, HtmlSelectOneRadio/ HtmlSelectOneListBox en HtmlDataTable. Deze worden in HTML weergegeven als een form, een input item, een radio group of een select list (drop-down) en een tabel.

Aan de slag met JSF in JDeveloper

Ter voorbereiding moet je eerst JDeveloper 10.1.3(.x) downloaden van OTN () en de zip-file uitpakken. Start dan de file JDEVELOPER_HOME\jdev\bin\jdev.exe. JDeveloper wordt nu gestart. Om een nieuwe applicatie te beginnen, ga naar File, New (of doe Ctrl+N) en kies de optie General, Applications en in het rechterscherm Application. In het pop-up window, geef de applicatie een naam en selecteer als Application Template de

optie: Web Application [JSF, ADF BC]. JDeveloper creëert de nieuwe Application met twee projecten: ViewController en Model. Negeer het Model project, voor nu is alleen het ViewController project interessant. JDeveloper heeft onder water een aantal webapplicatie-configuratiefiles aangemaakt, een JEE standaard-directorystructuur opgezet en enkele library's aan het project toegevoegd. We zijn er nu klaar voor om een eerste JSF-pagina te ontwikkelen.
Selecteer het ViewController project en ga opnieuw naar File, New, In the New Gallery, kies links de Web Tier en daaronder de JSF node. Selecteer dan rechts the JSF JSP-keuze. We gaan een JSP-pagina gebruiken als 'drager' voor de JSF componenten; dit is de meest gebruikelijke manier om JSF-pagina's te ontwikkelen voor een HTML-webapplicatie. Er verschijnt een wizard die naast de naam van de pagina nog een aantal details van ons wil weten. Accepteer alle defaults en druk op Finish (of Afsluiten als je de Nederlandse taalinstelling gebruikt). Nu wordt onze nieuwe pagina geopend in de WYSIWYG JSP editor.
Een JSF-pagina wordt opgebouwd als een boomstructuur, startend bij een <f:view> element. Alle UI-componenten die we gebruiken komen in deze boom te hangen. In de JSP kunnen we overigens 'gewone' JSP-tags, JSF-tags en gewone HTML-elementen gebruiken, hoewel het voordelen heeft uitsluitend met JSF-componenten te werken. In JDeveloper kunnen we razendsnel een pagina opbouwen door simpelweg de componenten van het JSF Componenten Palet op de JSP-editor (of het Structure Window) te slepen. Bij veel componenten zal er een simple pop-up scherm verschijnen waarin we property's voor de component – zoals id, prompt, lengte – kunnen invoeren.
In Figuur 1 zien we een pagina in aanbouw. Het is een boekenzoeker, waar de gebruiker enkele boekzoek-criteria kan opgeven. De pagina bevat een HtmlSelectManyListBox-component voor de Categorieën die als een HTML Select item wordt 'gerenderd' dat het multiple property op true heeft staan.



Figuur 1. Ontwikkel de JSF-pagina als een boom opgebouwd met UI Componenten, uit de Component Palette gesleept naar de Structure Window, de Source View of de WYSIWYG JSP editor.

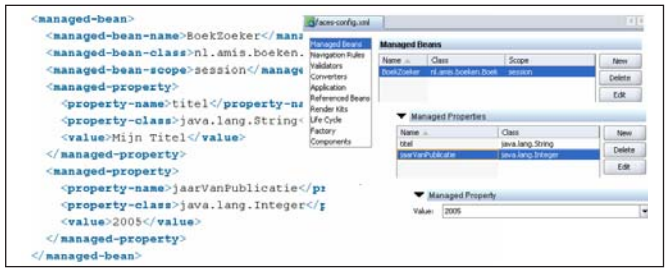
Overigens is het onwaarschijnlijk eenvoudig om van die representatie te schakelen naar een setje checkboxes door de selectManyListBox te vervangen door een selectManyCheckBox.
Een belangrijk kenmerk van JSF is dat de pagina-boom niet alleen wordt gebruikt om HTML te genereren. De boom en alle componenten zijn ook programmatisch toegankelijk voor onze applicatie. Net zoals in Oracle Forms – met de built-ins get_item_property() en set_item_property() – kunnen we de component lezen en manipuleren. Dynamisch de pagina herorganiseren, componenten op run-time toevoegen, property-waarden wijzigen: het kan allemaal!

Component eigenschappen

UI-componenten worden met property's geconfigureerd. Sommige property's zijn generiek en van toepassing op alle componenten, zoals id, value, rendered, binding, terwijl andere specifiek zijn. Alle Input Componenten kennen bijvoorbeeld de property's required, valueChangeListener, readOnly, validator, converter.
Een derde categorie property's zijn eigenlijk helemaal geen JSF-property's: de zogenaamde HTML pass-through property's zoals accesskey, title, maxlength, tabindex, styleClass (voor het HTML class-attribuut) en de event triggers voor change-, focus-, blur-, keydown- en andere events. Dit zijn eigenschappen die een betekenis hebben in een HTML-context en door de UI-componenten ongewijzigd worden doorgegeven in de gerenderde HTML. Ook deze property's kunnen overigens programmatisch gelezen en gemanipuleerd worden. In de volgende paragraaf zullen we zien hoe we property's niet alleen een statische waarde kunnen geven, maar ook een dynamisch te evalueren expressie. Dit is een cruciale factor in JSF!

Het model in JSF

Vrijwel alle property's van JSF Componenten kunnen worden gespecificeerd met behulp van expressies in de JSF Expression Language (EL). Deze JSF EL lijkt sterk op de JSP Expression Language die wordt gebruikt in onder meer JSTL (de JSP Standard Tag Library). JSF EL is op twee manieren rijker: het ondersteunt niet alleen lezen van, maar ook schrijven naar achterliggende objecten. Expressies kunnen ook methodes aanroepen naast property's benaderen. Een JSF EL expressie ziet er uit



Figuur 2. Managed Beans en de faces-config.xml Console.

als #{expressie}. De expressie kan operatoren gebruiken zoals +-*/ = empty ! true false, statische waarden als getallen en strings en verwijzen naar Java Beans.
Als een EL-expressie met verwijzing naar een bean wordt geëvalueerd kijkt JSF in alle 'context scopes', zoals de Servlet ApplicationScope, SessionScope en RequestScope. We hebben op die manier ook directe toegang tot request-parameters en applicatie configuratie-gegevens. JSF EL Expressies zien er bijvoorbeeld zo uit:

```
value="#{param.titel}"
rendered="#{param.jaarVanPublicatie > 1999}"
action="#{boekZoeker.doZoek}"
```

Het action property is verbonden met de doZoek() methode op de boekZoeker bean. Als de knop of link waar deze property mee verbonden is, wordt geactiveerd, roept JSF automatisch deze methode aan.

Managed Beans

In het recente verleden waar wij als ontwikkelaars volledig verantwoordelijk voor het instantiëren van alle objecten die onze JSP's nodig hadden om data op te vragen of op te slaan. JSF biedt meer geavanceerde faciliteiten en kan ons het beheer van de objecten uit handen nemen. Dit heet de Managed Beans functionaliteit (die overigens sterkt lijkt op de BeanContainer van het Spring Framework). In de faces-config.xml configuratie-file die JDeveloper heeft aangemaakt in de WEB-INF directory van de webapplicatie kunnen we onze Managed Beans configureren. Per bean specificeren we in elk geval de naam en scope van het object en ook de Java-class die wordt gebruikt om de bean te instantiëren. Bean in dit verband betekent niets meer dan een Java-object met een constructor die geen parameters nodig heeft. Via Managed Bean faciliteit kunnen we ook de initiele waarden van de bean property's vastleggen en ook referenties tussen de beans definiëren.
De Managed Bean definities worden vastgelegd in eenduidig maar wel tamelijk omvangrijk XML. JDeveloper heeft –zoals de meeste IDE's – een prettige user interface voor het beheer van onder meer de managed beans in de faces-config.xml file.

Het volgende code-fragment toont diverse property's in onze BoekZoek pagina die met EL-expressies gekoppeld zijn aan property's en methodes van managed beans:

```
<h:panelGrid columns="2">
          <h:outputText value="#{BoekZoeker titelPrompt}"
title="#{BoekZoeker.titelBubble}"/>
          <h:inputText id="titel" title="Boekentitel"
                                value="#{BoekZoeker.titel}"/>
          <h:outputText value="Jaar van publicatie"/>
          <h:inputText id="jaarVanPublicatie"
                                title="Jaar waarin het boek oorspronkelijk is verschenen"
                                value="#{BoekZoeker.jaarVanPublicatie}"/>
          <h:commandButton value="Zoek Boeken" action="#{BoekZoeker.doZoek}"/>
```

Zie bijvoorbeeld hoe het pass-through property title (het HTML property voor de tekst die boven komt drijven als je met de muis over een element beweegt en dat verder geen JSF betekenis heeft en direct wordt doorgegeven aan de browser) is verbonden met het titlebubble property on bookSearch. Het action property van de command button refereert naar de doZoek() methode.
Zoals eerder genoemd ondersteunt JSF EL tweerichtingsverkeer: waarden kunnen van beans worden gelezen en naar beans worden geschreven. Als bovenstaande paginafragment door JSF in HTML wordt vertaald, worden de huidige waarden van de titel en jaar van publicatie property's van de BoekZoeker bean in de corresponderende velden geschreven. De gebruiker kan vervolgens de waarden in het scherm wijzigen. Als de ZoekBoeken-knop dan wordt ingedrukt zorgt JSF er dankzij de EL-expressies in de value-property's voor dat de veldwaarden uit het HTML Form weer teruggeschreven worden naar de bean-property's. Hier is geen enkele inspanning van de ontwikkelaar voor nodig!

Events en Listeners

Je zou een Oracle Forms-ontwikkelaar 'trigger-happy' kunnen noemen: het is zo eenvoudig in Forms om kleine brokjes PL/SQL-code te associëren met events die in het Form plaatsvinden als gevolg van de acties van de gebruiker zoals het navigeren langs items en blokken, veranderen van veldwaarden, het indrukken van knoppen en zelfs het bewegen van de muis. JSF heeft een vergelijkbaar model: UI-componenten zoals Button en CommandLink vuren ActionEvents af als ze geactiveerd (aangeklikt) worden en alle Input Components publiceren ValueChangeEvents als hun waarde gewijzigd heeft. Het is overigens – anders dan in Forms – belangrijk om te realiseren dat hoewel de events door een gebruikersactie in browser ontstaan, de afhandeling van het event plaatsvindt in de server als de pagina gesubmit is en de componenten er achterkomen dat het event überhaupt heeft plaatsgevonden.
Zoals gezegd heeft iedere Input Component een valueChangedListener property dat gekoppeld wordt aan een methode in een of andere bean. Deze methode moet geen waarde terugggeven (void) en moet één inputparameter accepteren van het type ValueChangeEvent. Bijvoorbeeld:

```
public void auteurWijziging(ValueChangeEvent valueChangeEvent) {
 FacesContext fc = FacesContext.getCurrentInstance();
 fc.addMessage( valueChangeEvent.getComponent().getClientId(fc)
               , new FacesMessage("Waarde van auteur is gewijzigd. De
oude waarde was: "
                                  + valueChangeEvent.getOldValue())
               );
}
```

Deze valueChangeListener – min of meer de JSF-tegenhanger van de WHEN-VALIDATE-ITEM trigger, hoewel we specifiek voor validatie een andere JSF-feature gebruiken - schrijft een message naar de FacesContext als de waarde van het Auteur-veld wijzigt. De listener wordt als volgt geconfigureerd in de auteurscomponent:

```
<h:inputText id="auteur" value="#{boekZoeker.auteur}"
             valueChangeListener="#{boekZoeker.auteurWijziging }"/>
```

Je kunt overigens meerdere valueChangeListeners koppelen aan een inputComponent door valueChangeListener kind-elementen te gebruiken naast het property valueChangeListener. Buttons en Command Links genereren action events als ze geactiveerd worden. Action events triggeren ActionListeners. Die bestaan in drie verschijningsvormen: ActionListener child elementen, het actionListener property dat gekoppeld is aan een methode op een bean die ActionEvent als inputparameter krijgt en niets teruggeeft en het action property dat of een statische String bevat, een EL Expressie die een String oplevert of een bean methode aanroept die een String teruggeeft en geen inputparameters heeft. We zien later in dit artikel hoe die String een belangrijke rol speelt in de navigatie. De Zoek Boeken knop in onze voorbeeldapplicatie is verbonden met de doZoek() methode in de bean:

```
  <h:commandButton value="Zoek Boeken" action="#{boekZoeker.doZoek}"/>
<< einde kader met computercode >>

Dit is de doZoek() methode:

<< begin kader met computercode >>
    public String doZoek() {
        FacesContext fc = FacesContext.getCurrentInstance();
        fc.addMessage(null, new FacesMessage("Knop Zoek Boeken was
ingedrukt!"));
        return "zoek";
    }
```

Als we deze eenvoudige applicatie uitvoeren zien we een aantal interessante mechanismen aan het werk:

De velden Titel en JaarVanPublicatie hebben de initiële waarden die in de ManagedBeans sectie van de faces-config.xml file zijn gespecificeerd. Als ik een nieuwe waarde invoer in het Auteur-veld en dan de Zoek Boeken knop indruk, wordt de pagina ververst en verschijnen er twee meldingen: één vanwege het ValueChangeEvent dat door de listener op het auteur-veld is afgehandeld en de andere vanwege het ActionEvent dat is ontstaan door het indrukken van de Zoek Boeken knop. De waardes die ik in het scherm had ingevoerd zijn nu op de BoekZoeker bean vastgelegd. Dit klinkt misschien niet spectaculair, maar voor Java-webapplicaties hebben we al een heleboel voor elkaar gekregen op een heel eenvoudige en gestructureerde manier!

Converters en Validators

Als de gebruiker waarden invoert in het browser-formulier en dan de zoeken-knop indrukt, wordt het html-form met alle ingevoerde waarden gepost naar de applicatieserver. Wat de gebruiker ook invoert: getallen, tekst, datumwaarden, bedragen – alle waarden worden als strings verstuurd door de browser! Tegelijkertijd zijn de property's van de backing beans waar de velden uiteindelijk mee verbonden zijn 'strongly typed' – die zijn niet allemaal van het type String, maar van specifieke datatypes zoals Date, Integer, Long of applicatiespecifieke types als Postcode en Adres. Ergens in de verwerking van het browser-request moet iemand het op zich nemen om een conversie uit te voeren, van de Strings naar de specifieke types die corresponderen met de bean-property's. Binnen JSF zijn het de Converters die dat doen. Die zorgen er trouwens ook voor dat tijdens het genereren van de pagina de waardes die in hun specifieke type zijn vastgelegd op een nette manier geformatteerd naar de browser worden gestuurd, bijvoorbeeld java.util.Date weergeven als dd/mm/yyyy geformatteerde string.

Figuur 3. Run de Zoek Boeken pagina – Property binding en Events Listeners in actie.

JSF heeft ingebouwde converters voor veelvoorkomende types als Date en Number. Daarbij kan je specifieke formatteerpatronen meegeven, vergelijkbaar met de tweede parameter van de TO_CHAR, TO_DATE en TO_NUMBER functies in Oracle SQL. We kunnen ook prima onze eigen Converter schrijven, voor onze eigen types. Meer daarover in het derde artikel in deze serie. Overigens is JSF tamelijk flexibel: als je niet expliciet een converter specificeert kiest JSF zelf de best passende. Dat betekent in de praktijk dat je je voor verreweg de meest gebruikte types (String, Long, Integer, Date) helemaal niet over conversie hoeft na te denken.

Validatie van de invoer

Uiteraard zijn we als ontwikkelaars dol op onze eindgebruikers. Maar we weten dat we ze niet volledig kunnen vertrouwen. Of in elk geval dat ze wel eens een typefoutje maken bij het invoeren van gegevens in de applicatie. Om de kans op foutieve data in onze applicatie en zeker in de database te minimaliseren (op een voor de eindgebruiker aangename manier), implementeren we normaal gesproken datavalidaties. Java Server Faces heeft een ingebouwd validatie-mechanisme dat waarden controleert in de web-tier, voordat ze aan het model (de backing beans) worden doorgegeven. JSF heeft geen standaard ingebouwde client-side (JavaScript) validatie. Veel JSF-library's, waaronder ADF Faces, bieden die vorm van validatie wel.
De JSF validatie-infrastructuur gebruikt validator-elementen en validator-methodes. Een Validator is een generieke component die een specifieke, vaak enigszins te configureren validatie uitvoert voor de waarde ingevoerd in de UI Component waar de Validator aan gekoppeld is. JSF heeft ingebouwde Validators voor simpele validaties als LongRange die afdwingt dat numerieke waarden in een bepaald bereik liggen en Length die controleert of een ingevoerde waarde tenminste x en niet meer dan y lang is.
In onze simpele applicatie hebben we gespecificeerd dat het jaar van publicatie tussen 1998 en 2006 moet liggen. Als de titel als zoekcriterium wordt gebruikt, moet de ingevulde waarde tenminste vijf karakters bevatten:

```
<h:inputText id="titel" title="Boekentitel" value="#{BoekZoeker.titel}">
              <f:validateLength minimum="5"/>
</h:inputText>
<h:inputText id="jaarVanPublicatie"
             title="Jaar waarin het boek oorspronkelijk is verschenen"
             value="#{BoekZoeker.jaarVanPublicatie}">
   <f:validateLongRange maximum="2006" minimum="1998"/>
</h:inputText>
```

JSF laat ons ook eigen validators ontwikkelen, herbruikbare componenten voor veelvoorkomende dataregels. In plaats daarvan kunnen we ook Validation Method Bindings gebruiken, die een input-component koppelen aan een speciale validatiemethode op een managed bean, die er als volgt uit ziet:

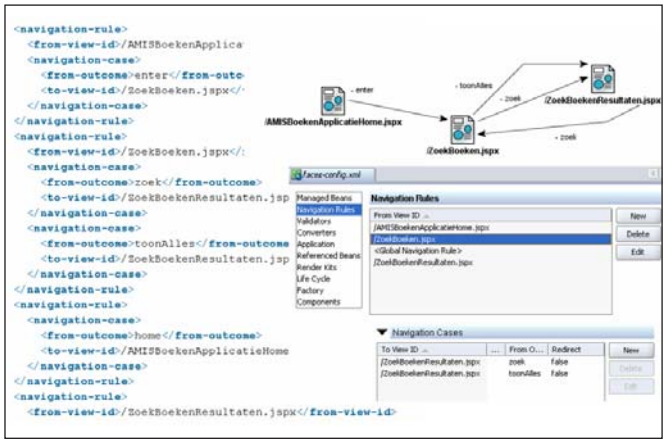
```
public void onderwerpenValidator(FacesContext facesContext,
                                 UIComponent uiComponent, Object
object) {
  if (((String[])object).length > 3) {
     ((UIInput)uiComponent).setValid(false);
     FacesMessage message =
         new FacesMessage("Het maximum aantal onderwerpen is 3!");
     facesContext.addMessage(uiComponent.getClientId(facesContext),
message);
        }
    }
```

Deze methode controleert of in de Input Component die wordt gevalideerd niet meer dan drie elementen geselecteerd zijn. De methode is gekoppeld aan de keuzelijst voor Onderwerpen:

```
<h:selectManyListbox  id="onderwerp" validator="#{BoekZoeker.onderwer-
penValidator}">
```

Navigatie

Tot dusverre bestaat onze applicatie uit één enkele pagina. Lekker overzichtelijk, maar niet erg realistisch voor een serieuze applicatie. Een belangrijke taak in een multi-pagina applicatie is de afhandeling van de navigatie tussen de pagina's. Hoe wordt navigatie geïnitieerd en waar wordt de gebruiker naar toe geleid. Daarbij volgen we het MVC (Model-View-Controller) patroon dat voorschrijft dat pagina's niet van elkaars bestaan weten. De navigatie-logica mag niet in een individuele pagina zijn vastgelegd, maar centraal in de Controller-component van de applicatie.
De centrale repository van Navigation Rules is de faces-config.xml file. Navigation rules beschrijven waar, komend vanaf een bepaalde pagina (aangeduid met de from-view-id) en gegeven de omstandigheden (de from-outcome), de applicatie de gebruiker heen voert (to-view-id). Figuur 4 toont de navigatieregels



Figuur 4. De navigatieregels (Navigation Rules) voor de boekenapplicatie.

voor onze applicatie – uitgebreid met een Welkom-pagina en een pagina met de zoek-resultaten, zowel in de faces-config.xml file als in de Faces Config console in JDeveloper.

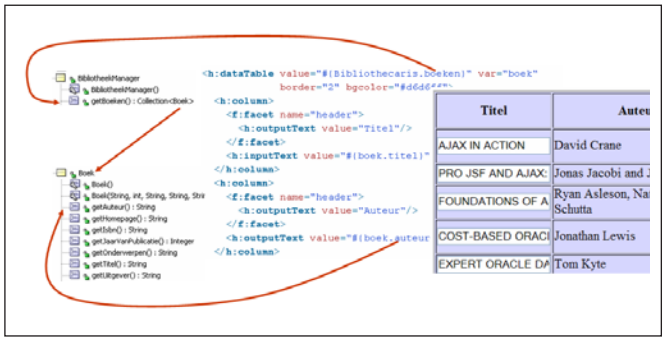
De vraag die nu bij je zou moeten opkomen is: waar komt die from-outcome waarde vandaan? Waar wordt die bepaald? Misschien heb je het al geraden: het action property dat we hebben gedefinieerd op button- en command-linkelementen bepaalt deze waarde, direct – een hard gecodeerde string – of indirect door een action method zoals de doZoek() methode aan te roepen, die een string retourneert.

Nog meer

Hiermee hebben we de fundamenten van JSF besproken. Er is natuurlijk nog veel meer over te zeggen: een gemiddeld JSF-boek heeft zeker niet minder dan 500 bladzijden. Onderwerpen als Resource Bundles en internationalisatie, AJAX, JSF voor andere views dan HTML en programmatisch manipuleren van UI-componenten laten we noodgedwongen achterwege. Maar met de bouwstenen die we hierboven besproken hebben, kan je aan de slag. In de volgende twee artikelen gaan we dieper in op een aantal aanvullende onderwerpen.

Aantrekkelijk productief

Tot nog toe hebben we wel erg simpele en sobere schermen gezien. Met een paar leuke plaatjes en een CSS-stylesheet hadden we de boel al flink kunnen opfleuren, al had dat nog geen functionaliteit toegevoegd. Laten we nog snel even kijken naar verreweg de meest interessante en complexe component in de JSF standaard: de HtmlDataTable.
Deze component toont een collectie objecten in een tabelletje. Hij ondersteunt Row- en Column-banding (verschillende kleuren voor opvolgende rijen en kolommen), een header en footer element, paginering (toon rijen vanaf element x met maximaal y elementen) en (multi-record) edit (!). De DataTable kan gekoppeld worden aan een Java Collectie van Beans, een Array en ook een JDBC ResultSet – dus rechtstreeks aan een database query! JDeveloper opent een wizard als we de DataTable van het palet naar de pagina slepen. In de wizard kunnen we de tabel associëren met een methode die een Collectie teruggeeft. De wizard gebruikt introspectie om de property's van de beans in de collectie te bepalen en komt op basis daarvan met een voorstel voor de kolommen in de tabel. Via deze wizard kan je binnen een minuut een geavanceerde tabelcomponent introduceren in een pagina.
Onze Boeken Applicatie heeft een managed bean bibliotheekManager. Deze bean heeft een methode getBoeken() die een collectie van Boek objecten teruggeeft. De Boek-class heeft voor de hand liggende property's als titel, ISBN, auteur en jaar van publicatie. Door simpelweg de DataTable met deze methode te verbinden en de wizard de kolommen te laten associëren met de property's van de impliciete boekiterator van het type Boek, kunnen we de DataTable activeren. Figuur 5 illustreert dit. Om de tabel van read-only in edit-mode te krijgen hoeven we slechts de outputText elementen te veranderen in inputText elementen!



Figuur 5. Toepassing van de DataTable component op basis van een Collectie van Boeken met de Boek property's in de kolommen.

Vergeleken met de gebruikelijke Java-webapplicatieontwikkeling is deze tabel-component en de manier waarop je die in JDeveloper zonder programmeren kan toepassen werkelijk een openbaring. Het lijkt de Oracle Forms datablock-wizard wel! In deel twee gooien we er nog een schepje bovenop, met ADF Faces. Dan komen we heel dicht bij de productiviteit en het databasegerichte ontwikkelgemak van Oracle Forms.

Referenties

- *The Complete Reference: Java Server Faces* – Chris Schalk & Ed Burns, McGrawHill/Osborne, 2006 ISBN 0072262400
- *Java Server Faces in Action* – Kito D. Mann, Manning Publications, 2004, ISBN 1932394125
- *JSF Central* - http://www.jsfcentral.com/ - website voor de JSF ontwikkelaarsgemeenschap met bergen links naar JSF sites, tools, artikelen etc.
- *AMIS Technology Weblog* – http://technology.amis.nl/blog - tientallen artikelen en voorbeelden van JSF
- *Workshop Pretty Java Server Faces* – een introductie op JSF (Nederlandstalig) http://www.amis.nl/activiteiten.php?id=316
- *Sources voor dit artikel (JDeveloper 10.1.3.1 Application en Projecten)*: http://www.amis.nl/tech_artikelen.php?id=409

Lucas Jellema is Technology Manager bij AMIS. Daarnaast schrijft hij artikelen op de AMIS Weblog (http://technology.amis.nl/blog) en in tijdschriften als Optimize, Java Magazine en Java Developer Journal. Hij is zowel Oracle ACE als Oracle Regional Director for Fusion Middleware. Lucas kan bereikt worden via email: jellema@amis.nl.