Sinds dit voorjaar is er een nieuwe standaard, Java Server Faces (JSF), voor het ontwikkelen van Java webapplicaties. JSF heeft als fundamenten de aloude servlet en JSP specificaties en biedt een Web Applicatie Framework gebaseerd op het Model View Controller (MVC) design pattern. In dit artikel gaat Willem Koppenol in op de onderdelen en opbouw van een JSF applicatie, aan de hand van een simpele login applicatie.

thema

Java Server Faces: flexibele standaard

Introductie event driven programmeer model

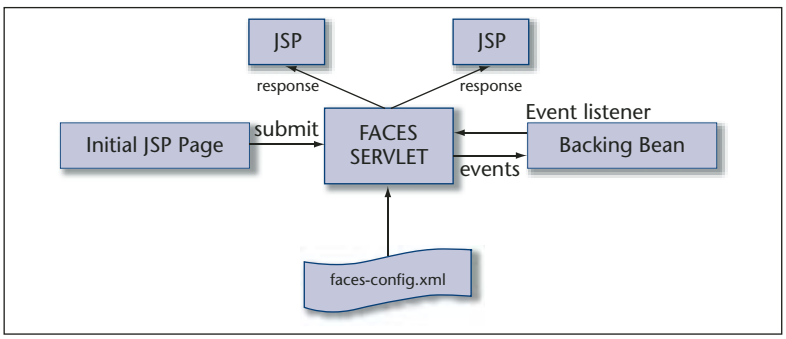
Met JSF heeft de webontwikkelaar de beschikking over onder meer een reeks standaardcomponenten, een uitbreidbaar componenten model, tag library's voor interactie met JSP, navigatie-ondersteuning, validatie- en conversiecomponenten, gestandaardiseerde foutafhandeling en ingebouwde ondersteuning voor internationalisatie. De belangrijkste innovatie is echter dat met JSF het event driven programmeer model, zoals dat bekend is van tools als Visual Basic en Delphi, in de Java webapplicatie-wereld wordt geïntroduceerd. JSF is daarmee een antwoord op Microsoft's ASP.NET dat dit programmeermodel al eerder voor .NET webapplicaties introduceerde. Een andere belangrijke inspiratiebron van JSF is het open source Struts Framework, waarvan het navigatiemodel in grote lijnen door JSF is overgenomen.

Model View Controller JSF is gebaseerd op de Model View Controller (MVC) architectuur. De Controller is bij JSF een centraal `FacesServlet` dat de navigatie beslissingen in de webapplicatie neemt. Alle requests in een JSF applicatie lopen via dit centrale `FacesServlet`. De navigatieregels zijn vastgelegd in het `faces.config.xml` bestand dat door het `FacesServlet` wordt gelezen. Het centrale `FacesServlet` delegeert volgens de MVC architectuur het uitvoeren van business-logica en het benaderen van data aan de Model componenten (Java beans). De output naar de client browser wordt gedelegeerd aan de View componenten (JSP pagina's).

Een JSF applicatie heeft dezelfde structuur als een standaard Java Web Applicatie. Een centrale plaats wordt ingenomen door de deployment descriptor `web.xml` in de `WEB_INF` directory. Deze wordt bij het opstarten door de Web Server gelezen en bevat declaraties van servlets die geactiveerd moeten worden en initialisatie bestanden die moeten worden gelezen. Bij onze login applicatie staat in `web.xml`:

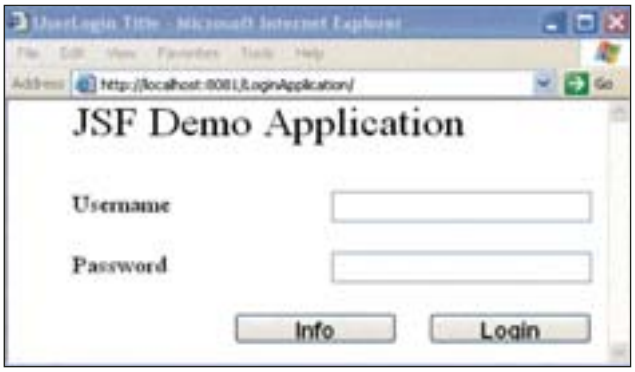
```
<servlet>
   <servlet-name>Faces Servlet</servlet-name>
   <servlet-
   class>javax.faces.webapp.FacesServlet</ser-
   vlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```



FIGUUR 1. JSF Overview

Adv. Compuware



FIGUUR 2. Login scherm

Het servlet-mapping element geeft aan dat alle requests met het URL patroon /faces/ door het FacesServlet worden afgehandeld.

UI COMPONENTEN Een belangrijk onderdeel van JSF is het User Interface (UI) Component Framework. Deze UI componenten leven op de server, worden getoond aan de client en kunnen hun toestand tussen client requests in vasthouden. Onder de standaard UI componenten bevinden zich oude bekenden die overeenkomen met HTML tags, zoals labels, tekst velden, hyperlinks, buttons en list boxen. Maar er zijn ook geavanceerdere componenten waarmee bijvoorbeeld data binding mogelijk is of die foutmeldingen ophalen en tonen. In onze login applicatie hebben we een startpagina met labels, knoppen, een tekst en een password veld. Niet zichtbaar is een message component waarin eventuele foutmeldingen worden getoond. De Info knop toont de gebruiker de geldige username en password combinatie en met de login knop wordt daadwerkelijk ingelogd.

TAG LIBRARIES De UI componenten worden in JSP pagina's gerepresenteerd door JSF custom tags. JSF kent een tweetal custom tag libraries. De component tag library definieert de tags die de UI componenten representeren. De core tag library definieert tags die het mogelijk maken events te registreren, en validators en converters aan componenten te koppelen. De custom tags komen door de volgende directives in een JSP pagina beschikbaar:

```
<%@ taglib uri="http://java.sun.com/jsf/html"
prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core"
prefix="f" %>
```

Met opname van deze taglib directives ziet de achterliggende JSP code met JSF tags van onze login pagina er als volgt uit:

```
<f:view>
  <h:form binding="#{UserBean.form1}"
  id="form1">
   <h:outputText binding="#{UserBean.titLbl}"
   id="titLbl" " value="JSF Demo"/>
   <h:outputText binding="#{UserBean.usLabel}"
   id="usLabel" value="Username"/>
   <h:outputText binding="#{UserBean.pwLabel}"
   id="pwLabel value="Password"/>
   <h:inputText binding="#{UserBean.userName}"
   id="userName" required="true"/>
   <h:commandButton
   action="#{UserBean.login_action}"
          binding="#{UserBean.loginButton}"
          id="loginButton"  value="Login"/>
   <h:commandButton
   action="#{UserBean.info_action}"
          binding="#{UserBean.infoButton}"
          id="infoButton"  value="Info"/>
   <h:inputSecret
   binding="#{UserBean.passWord}"
   id="passWord"
          validator="#{UserBean.length-
          Validator1.validate}"/>
    <h:messages binding="#{UserBean.mList1}"
    id="mList1" showDetail="true"/>
  </h:form>
</f:view>
```

BACKING BEANS Wat opvalt aan bovenstaande code zijn de binding attributen van de verschillende UI componenten. De expressie #{UserBean.userName} is Java Expression Language (EL) die het gelijknamige tekst veld koppelt aan de userName property van de bean UserBean. Hierdoor worden de gegevens in de invoervelden op het scherm automatisch gesynchroniseerd met de property's van een UserBean op de server. Deze beans worden daarom backing beans genoemd. Na een geslaagde login bevat de bean de userName en passWord combinatie. Bij fouten worden reeds ingevulde data vanuit de bean opnieuw op de pagina gezet. Voor ontwikkelaars die bekend zijn met Struts zal dit bekend voorkomen want het lijkt erg op de functionaliteit van ActionForm classes. Overigens kan de koppeling behalve met het binding attribuut ook met het value attribuut worden gelegd. In dat geval is er geen koppeling met de UI Component maar alleen met de waarde in de UI Component. Een gedeelte van de code van de login bean ziet er als volgt uit:

```
public class UserBean extends
AbstractPageBean {
   private HtmlInputText userName = new
   HtmlInputText();
   public HtmlInputText getUserName() {
```

```
        return userName;
    }
    public void setUserName(HtmlInputText hit)
    {
        this.userName = hit;
    }

    private HtmlInputSecret passWord = new
    HtmlInputSecret();
    public HtmlInputSecret getPassWord() {
        return passWord;
    }
    public void setPassWord(HtmlInputSecret
    his) {
        this.passWord = his;
    }
}
```

Het is in JSF niet perse noodzakelijk om één bean te gebruiken voor iedere UI pagina. Een pagina kan ook worden gekoppeld aan meer dan één bean. Verder introduceert JSF het concept van de managed beans. Deze beans worden in het centrale configuratie bestand, `faces-config.xml`, gedeclareerd. Ze worden dan automatisch gecreëerd als ze de eerste keer worden benaderd. Creatie door middel van Java code of JSP tags is niet nodig. De scope van de managed beans kan ook worden aangegeven in `faces-config.xml` en varieert tussen request, session of applicatie:

```
<managed-bean>
  <managed-bean-name>UserBean</managed-bean-
  name>
  <managed-bean-
  class>jsfDemo.UserBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-
  scope>
</managed-bean>
```

EVENTS De belangrijkste innovatie van JSF is het event driven programmeer model. Het drukken op een knop in een JSF webpagina kan eenvoudig gekoppeld worden aan het uitvoeren van een functie in een backing bean op de server. Het gehanteerde event model is het standaard Java publish en subcribe event model. UI componenten genereren events na activering door de gebruiker. Deze events worden opgevangen door event listeners die zich bij de UI Componenten op de events hebben geabonneerd.

JSF onderscheidt een tweetal standaard events. Het `ActionEvent` gaat af als UI componenten die het `ActionSource` interface implementeren, zoals buttons en hyperlinks, worden geactiveerd. Abonnees implementeren het `ActionListener` interface met de `processAction` methode. Het `ValueChanged` event gaat af als de gebruiker de waarde van `UIInput` component of één van de daarvan afgeleide classes wijzigt, zoals invoervelden, check en list boxen. De abonnees op deze events implementeren het `ValueChangeListener` interface met de `processValueChange` methode. Beide methodes krijgen het event object als parameter mee om eventuele nadere gegevens uit te lezen.

Een applicatie kan op een tweetal manieren reageren op gegenereerde events. Ofwel de listener class wordt bij de UI Component geregistreerd door een `valueChangedListener` tag of een `actionListener` tag op te nemen onder de UI Component tag. Ofwel een methode uit de backing bean wordt door een binding expressie gekoppeld aan het juiste attribuut, i.e. `action`, van de UI Component tag. Dit laatste is het geval bij de Login en Info buttons in de login pagina en ziet dit er als volgt uit :

```
<h:commandButton
action="#{UserBean.login_action}"
          binding="#{UserBean.loginButton}"
          id="loginButton"  value="Login"/>
<h:commandButton
action="#{UserBean.info_action}"
          binding="#{UserBean.infoButton}"
          id="infoButton"  value="Info"/>
```

De implementatie van de `info_action` event functie in de backing bean toont een nieuwe webpagina met de benodigde login informatie. De `login_action` event functie controleert de login gegevens en toont dan ofwel een welkomst scherm met de gebruikers naam of een hernieuwd login scherm met een foutmelding:

```
public java.lang.String login_action() {
    // Add your event code here...
    String uname = (String)userName.get-
    Value();
    String pword = (String)passWord.get-
    Value();
    if ( uname.equals("JSFNovice") &&
pword.equals("JSF")) {
        return "success";
    }
    else {
        FacesContext context = FacesCon-
        text.getCurrentInstance();
        FacesMessage message = new
        FacesMessage("Invalid Username

                and/or Password");
        context.addMessage("form1", message);
        return "failure";
```

```
    }
}

public java.lang.String info_action() {
    return "info";
}
```

FIGUUR 3. Webpagina na klikken op de Info button

NAVIGATIE REGELS JSF biedt ook faciliteiten voor het vastleggen van de navigatie structuur van webapplicaties. De navigatie regels worden gedefinieerd in FACES-CONFIG.XML en koppelen strings aan URL's. Het FACES-CONFIG.XML bestand vormt het hart van iedere JSF applicatie en wordt door het `FacesServlet` bij het opstarten gelezen. Het bevat naast de definitie van de navigatieregels zoals we zagen ook de declaratie van de Managed Beans. Het `FacesServlet` gebruikt de navigatieregels om te bepalen wat er moet gebeuren als bepaalde acties plaatsvinden. Een voordeel van het vastleggen van de navigatie structuur in een centraal extern bestand is, dat het wijzigen van de navigatie eenvoudiger wordt. De navigatiestructuur kan in verschillende tools ook grafisch worden vastgelegd.

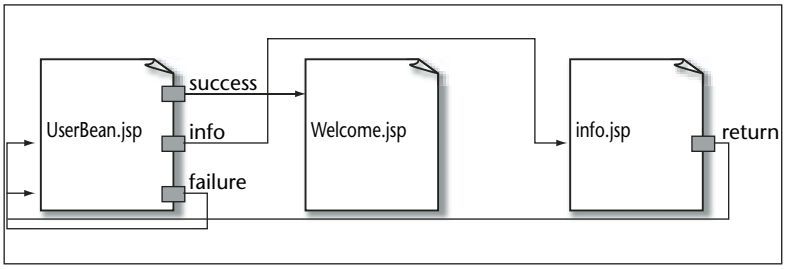
Het faces-config.xml bestand lijkt sterk op het struts-config.xml bestand uit Struts en is daar zeker door geïnspireerd. In onze login applicatie retourneert de login_action afhankelijk van de userName en passWord combinatie "success" of "failure". Het faces-config.xml bestand geeft dan aan welke pagina vervolgens zal verschijnen:

```
<faces-config>
    <navigation-rule>
        <from-view-id>/Info.jsp</from-view-id>
        <navigation-case>
            <to-view-id>/UserBean.jsp</to-view-
            id>
            <from-outcome>return</from-outcome>
        </navigation-case>
    </navigation-rule>
    <navigation-rule>
        <from-view-id>/UserBean.jsp</from-view-
        id>
        <navigation-case>
            <to-view-id>/Welcome.jsp</to-view-
            id>
            <from-outcome>success</from-outco-
            me>
        </navigation-case>
        <navigation-case>
            <to-view-id>/Info.jsp</to-view-id>
            <from-outcome>info</from-outcome>
        </navigation-case>
        <navigation-case>
            <to-view-id>/UserBean.jsp</to-view-
            id>
            <from-outcome>failure</from-outco-
            me>
        </navigation-case>
    </navigation-rule>
</faces-config>
```

VALIDATORS Een veel voorkomende en vaak tijdrovende taak in webapplicaties is het valideren van ingevoerde gegevens. Dit is vaak noodzakelijk, omdat het problemen voorkomt bij de verdere verwerking van de gegevens. Voor deze taak biedt JSF ondersteuning in de vorm van een reeks standaard-validatiecomponenten. Ontwikkelaars kunnen hiermee veel tijd besparen. De validatiecomponenten worden gekoppeld aan invoervelden en zorgen dat de invoer binnen bepaalde grenzen blijft. De `LengthValidator` beperkt bijvoorbeeld het maximaal in te voeren aantal karakters en de `LongeRangeValidator` en `DoubleRangeValidator` zorgen dat ingevoerde getallen binnen bepaalde grenzen blijven. De validatiecomponenten genereren in geval van ongeldige invoer een fout. Het `FacesServlet` zal vervolgens de oorspronkelijk pagina opnieuw tonen met zo mogelijk de foutmelding in een `Message` component. Bij de login applicatie is een `LengthValidator` op de volgende manier gekoppeld aan het invoerveld voor het `passWord` :

```
<h:inputSecret binding="#{UserBean.passWord}"
id="passWord"

validator="#{UserBean.lengthValidator1.valida-
te}"/>
```



FIGUUR 4. Grafische voorstelling van de navigatiestructuur in de login applicatie

Validatie met validatiecomponenten vindt plaats op de server. Desgewenst kan ook gevalideerd worden met Javascript op de client. De UI Componenten hebben daartoe property's die aan Javascript event functions kunnen worden toegekend. Overigens is het model voor validatiecomponenten in JSF precies gespecificeerd en het is dan ook mogelijk om je eigen validatiecomponenten te schrijven. Al wat dient te gebeuren is een implementatie te verzorgen van het `Validator` interface met de `validate` methode:

```
public class PasswordValidator implements
Validator {
public void validate(FacesContext context,
UIComponent component,
                     Object toValidate) {}
}
```

Naar verwachting zullen IDE's die JSF implementeren weldra met meer geavanceerdere validatie componenten op de markt verschijnen. Met name een reguliere expressievalidator, zoals ook aanwezig in ASP.NET, zou erg handig zijn.

ERROR HANDLING De `UIMessage` en `UIMessages` componenten worden in JSF gebruikt voor het tonen van foutmeldingen. Ze worden op een webpagina geplaatst door respectievelijk de `<message>` en de `<messages>` tags. Op de plek van de `<messages>` tag worden alle foutmeldingen getoond, terwijl de `<message>` tag een for attribute heeft om een koppeling te leggen met één specifieke component. Een `<message>` tag die een foutmelding toont als een invoer veld niet is ingevuld wordt als volgt gekoppeld:

```
<h:inputText binding="#{UserBean.userName}"
id="userName" required="true"/>
<h:message for="username" showDetail="true"/>
```

Voor het tonen van foutmeldingen bekijken de `UIMessage` componenten de `FacesContext`. De backing

FIGUUR 5. Foutmelding in een UIMessages Component

beans kunnen de `FacesContext` benaderen en er foutmeldingen als `FacesMessages` aan toevoegen. Het resultaat - een verkeerde login - is te zien in Figuur 5. De foutboodschap is in de `login_action` functie gegenereerd.

INTERNATIONALISATIE JSF ondersteunt de internationalisatie van alle meldingen in de applicatie door ze te definiëren in `ResourceBundles`. Een `ResourceBundle` bevat een reeks gelokaliseerde meldingen en wordt opgeslagen in een bestand met de extensie `.properties`. Voor de login applicatie heet dit bestand bijvoorbeeld `LoginMessages.properties` en bevat id's van meldingen en de vertaling daarvan in de oorspronkelijke taal:

```
username_label=Username
password_label=Password
login_button=Login
```

Om de meldingen in de `ResourceBundle` te kunnen gebruiken refereert een webpagina met JSF tags met een `loadBundle` tag aan de `ResourceBundle`:

```
<f:loadBundle
basename="message.LoginMessages" var="bund-
le"/>
```

Voor andere taal/land combinaties worden extra `.properties` bestanden toegevoegd met de taal/land combinatie als extra extensie. De applicatie refereert vervolgens in JSF tags aan de id's van de meldingen en de vertaling in de huidige taal land combinatie wordt ingevuld.

CONVERTERS JSF kent ook een standaardmechanisme om conversies toe te passen op in UI componenten ingevoerde gegevens. Deze gegevens zijn in de webpagina's van het type string. Door converters te koppelen aan UI componenten worden deze strings voor verdere verwerking geconverteerd naar het type van de converter. De koppeling wordt aangebracht door de converter als attribuut op te nemen bij de UI Component. Onder de standaardconverters die het JSF Framework levert zijn onder andere een `DoubleConverter`, een `BooleanConverter` en een `DateTimeConverter`. De converters kunnen ook een rol spelen bij het tonen van gegevens aan de gebruiker. De conversie verloopt dan andersom. Verder kunnen in JSF eenvoudig eigen converters worden geschreven door de Converter interface te implementeren. Deze interface bestaat uit de volgende twee methoden:

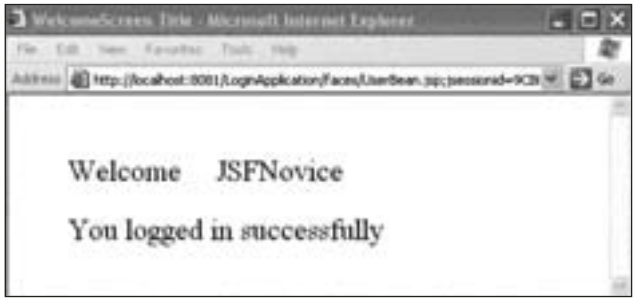
```
public class MyConverter implements Converter
{
public Object getAsObject(FacesContext con-
text, UIComponent component,
                                String
newValue) throws ConverterException {}

public String getAsString(FacesContext con-
text, UIComponent component,
                          Object value)
throws ConverterException {}
}
```

RENDERERS Het laatste kenmerk van JSF dat ik wil vermelden, heeft betrekking op de opmaak van componenten in JSF. Deze wordt overlaten aan aparte `Renderers`. Als in webpagina's de JSF component tag library wordt gebruikt, zijn de gebruikte UI componenten generiek en niet gebonden aan een specifieke opmaak taal. In een standaard JSF implementatie is een default HTML Renderer aanwezig die de pagina's in HTML opmaakt voor web browsers. Voor het hergebruik van de pagina's in mobiele telefoons, moet een aparte WML (Wireless Markup Language) Renderer worden gebruikt. De oorspronkelijke UI componenten kunnen ongewijzigd blijven. Het slotscherm van de login applicatie kan nu verschijnen. De naam van de succesvol ingelogde gebruiker wordt uit de property's van de backing bean gehaald (zie Figuur 6).

SLOTWOORD Een belangrijk voordeel van JSF ten opzichte van andere Java Web Application Frameworks is dat het een Java-standaard is en als zodanig zal worden opgenomen in J2EE. JSF is zeer flexibel en biedt enorm veel mogelijkheden. Aanvankelijk is de JSF standaard echter nogal sceptisch ontvangen door sommige Java-ontwikkelaars. Er was met name nogal wat kritiek op de keerzijde van de flexibiliteit, namelijk de complexiteit van de specificatie. JSF zal pur sang inderdaad voor de gemiddelde ontwikkelaar moeizaam in applicaties te gebruiken zijn. Dit is echter nooit de bedoeling geweest, zoals ook in de specificatie staat vermeld: "JSF is bedoeld voor fabrikanten van tools en componenten die zich met JSF kunnen focussen op een enkel component Framework voor Java Web Applicaties". De acceptatie van JSF zal afhangen van de implementatie in tools. Sun gaf onlangs het goede voorbeeld door met Java Studio Creator, voormalig project Rave, een eerste RAD en "ease of use" JSF ontwikkeltool te tonen. Alle andere toonaangevende toolfabrikanten hebben aangekondigd dit op korte termijn ook te doen. De verwachting is dat JSF daarna in rap tempo de wereld zal veroveren en we Java Web Applicaties kunnen bouwen door het draggen en droppen van componenten en het invoeren van event handling code. De achterliggende JSF tags zullen voor ons worden gegenereerd. Het is dan ook aan te raden om nieuwe Java Web Applicaties vanaf nu direct in JSF te bouwen. Graig McClanahan, grondlegger van Struts en specification lead van de JSR-127 beveelt aan om Struts-applicaties om te bouwen naar JSF in plaats van de twee te integreren. En er zijn al library's beschikbaar die deze migratie vergemakkelijken.

> JSF heeft het navigatiemodel van het open source Struts Framework in grote lijnen overgenomen



FIGUUR 6. De gebruikersnaam wordt uit de property's van de backing bean gehaald

drs. Willem Koppenol is Product Specialist Software Development Training bij Twice IT Training (e-mail: wkoppenol@twice.nl).