TOOLS

Een goed voorbeeld van een best practice in softwareontwikkeling is het principe van high cohesion van componenten en het principe van loose coupling van componenten. Het eerste houdt in dat iedere component een eigen, heldere verantwoordelijkheid heeft, terwijl het tweede impliceert dat alle componenten zoveel mogelijk onafhankelijk van elkaar zijn en alleen via stabiele interfaces interacten.

Facelets, een goede uitbreiding van JSF

Maakt implementatie UI-componenten simpeler

Jonck van der Kogel
is Softwareengineer en manager Java Competence Center bij HintTech.

Eelco Klaver
is Senior consultant bij E.Consulting.

Hierdoor wordt de onderhoudbaarheid, herbruikbaarheid en leesbaarheid van de code verbeterd. Een component blijft volledig verantwoordelijk voor de taken waarvoor het bedoeld is. Het is niet de bedoeling dat de code moet worden aangepast teneinde het component te kunnen gebruiken.

De voornaamste kracht van JavaServer Faces (JSF) is de mogelijkheid om user interface-componenten te ontwikkelen. Facelets is een uitstekende uitbreiding op JSF en maakt het eenvoudiger om herbruikbare user interface-componenten te implementeren voor op de JSF-pagina's. Voorkomende elementen binnen een webapplicatie kunnen als Facelets worden ingericht en vervolgens binnen de pagina als custom tag worden opgenomen. Eenvoudige Facelets zonder achterliggende Java logica kunnen aan een xhtml-pagina worden toegevoegd zonder dat deze hier voor aangepast hoeft te worden.
Het wordt anders op het moment dat een Facelet meer ingewikkelde logica bevat, zoals een tabel die op alle kolommen sorteerbaar is. In dat geval moet de sorteerlogica in een backing bean geïmplementeerd worden. We willen deze sorteerlogica echter niet in de backing bean van de pagina opnemen, omdat in dat geval de functionaliteit niet herbruikbaar is over meerdere pagina's.

Probleemstelling

Facelets voorziet op xhtml-niveau in user interface-componenten. Neem als voorbeeld een non-breaking space; om deze binnen JSF te realiseren, moet het volgende geschreven worden:

```
<f:verbatim> </f:verbatim>
```

Om dit overal binnen een applicatie uit te schrijven is lastig en foutgevoelig. Eenvoudiger is het om hiervoor een Facelet te schrijven:

```
<!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:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core">

<ui:composition>
    <f:verbatim> </f:verbatim>
</ui:composition>

</html>
```

Om dit als Facelet te gebruiken binnen een JSF-pagina, wordt de syntax:

```
<example:space/>
```

Bovenstaande is een eenvoudig maar ook krachtig voorbeeld van de kwaliteit van Facelets.
Bij een Facelet is het mogelijk om attributen door te geven op het xhtml-niveau. Als er een object meegegeven moet worden aan een Facelet wordt dat als volgt:

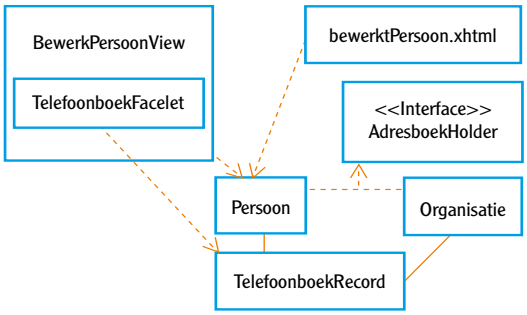
```
<example:facelet exampleAtribute="#{exampleBean}" />
```

De Facelet kan vervolgens van het attribuut "exampleAttribute" gebruik maken binnen de xhtml. Indien een Facelet een backing bean heeft, is het object "exampleAttribute" niet beschikbaar binnen de Java-code. Dus hoewel de Facelet op xhtml-niveau inzetbaar is als component zal de achterliggende Java-code van de pagina aangepast

moeten worden wanneer een pagina gebruik maakt van een Facelet.

Voorbeeld applicatie

Als voorbeeld voor een Facelet wordt er gebruik gemaakt van een zeer minimalistisch telefoonboek. De gedachte achter dit voorbeeld is dat vele entiteiten een telefoonboek kunnen hebben. Zo kan een webapplicatie verschillende pagina's bevatten waarbij voor een persoon en voor een organisatie een telefoonboek wordt bijgehouden. Aangezien het voor de presentatie van een telefoonboek niet uitmaakt in welke context het getoond wordt, kan het telefoonboek hergebruikt worden en zou het logisch zijn om een functionaliteit als Facelet uit te voeren.



De pagina (hierna de 'main view' genoemd) waar de Facelet gebruikt zal gaan worden heeft twee mogelijke toestanden: bewerken of bekijken. Er kan tussen deze twee toestanden worden gewisseld door de betreffende knop 'bewerken' of 'bewaren/annuleren'.

De main view heeft een backing bean, die op zijn beurt een Persoon-object bevat dat een lijst met telefoonboekrecords bijhoudt. De Facelet biedt de mogelijkheid om met een knop een nieuw telefoonboekrecord aan te maken die vervolgens twee nieuwe invoervelden toont. Main view:

```
<!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:ui="http://java.sun.com/jsf/facelets"
  xmlns:t="http://myfaces.apache.org/tomahawk"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:example="http://example.org"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:a4j="https://ajax4jsf.dev.java.net/ajax">

<head>
  <title>FaceletsArtikel</title>
</head>
<body>
<f:view>
  <h:form>
    <t:saveState value="#{exampleView.persoon}" />
    <t:saveState value="#{exampleView.editMode}" />
    <a4j:region>
      <t:div id="exampleView">
        <a4j:commandButton value="Bewerken"
          action="#{exampleView.edit}"
          rendered="#{not exampleView.editMode}"
          reRender="exampleView" />
        <a4j:commandButton value="Bewaren"
          action="#{exampleView.save}"
          rendered="#{exampleView.editMode}"
          reRender="exampleView" />
      <example:addressBook backingBean="#{exampleView}" />
        </t:div>
      </a4j:region>
  </h:form>
</f:view>
</body>
</html>
```

AddressBook:

```
<!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:ui="http://java.sun.com/jsf/facelets"
  xmlns:t="http://myfaces.apache.org/tomahawk"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:a4j="https://ajax4jsf.dev.java.net/ajax">

<ui:composition>
  <a4j:region>
    <a4j:commandButton value="Nieuw record"
          action="#{backingBean.nieuwRecord}"
          reRender="telefoonBoek"
          rendered="#{backingBean.editMode}" />
      <t:div id="telefoonBoek">
          <h:dataTable  value="#{backingBean.persoon.
           telefoonBoekRecords}"
           var="record">
          <h:column>
          <f:facet name="header">
          <h:outputText value="Naam" />
          </f:facet>
          <h:inputText value="#{record.naam}"
           rendered="#{backingBean.editMode}" />
          <h:outputText value="#{record.naam}"
           rendered="#{not backingBean.editMode}" />
        </h:column>
        <h:column>
          <f:facet name="header">
           <h:outputText value="Nummer" />
          </f:facet>
          <h:inputText value="#{record.nummer}"
            rendered="#{backingBean.editMode}" />
          <h:outputText value="#{record.nummer}"
            rendered="#{not backingBean.editMode}" />
        </h:column>
      </h:dataTable>
    </t:div>
  </a4j:region>
</ui:composition>
</html>
```

Zowel de backing bean van de main view als van de Facelet worden op request scope gedefinieerd. Er moet een methode komen waarmee een nieuw record aangemaakt wordt. Dit betekent dat op Java-niveau een nieuw telefooboekrecordobject moet worden aangemaakt. Deze wordt toegevoegd aan de lijst op het Persoon-object.

De vraag is nu waar we deze nieuwRecord() methode zouden moeten plaatsen. De nieuwRecord() methode zal er ongeveer als volgt uit moeten komen te zien:

```
public void nieuwRecord() {
        TelefoonBoekRecord record = new
TelefoonBoekRecord();
        adresBoekHolder.addTelefoonBoekRecord(record);
}
```

Er zijn drie plekken waar de nieuwRecord() methode geplaatst kan worden:

1. op het Persoon-object;
2. op de backing bean van de main view;
3. op de backing bean van de Facelet.

Het telefoonboek kan hergebruikt worden door gebruik te maken van een Facelet.

Main view bean met de nieuwRecord methode:

```
package faceletsArtikel.source;

public class ExampleView {
    private Persoon persoon;
    private boolean editMode;

    public ExampleView() {
        persoon = new Persoon();
    }

    public void nieuwRecord() {
        TelefoonBoekRecord record = new
TelefoonBoekRecord();
        persoon.addTelefoonBoekRecord(record);
    }

    public Persoon getPersoon() {
        return persoon;
    }

    public void setPersoon(Persoon persoon) {
        this.persoon = persoon;
    }

    public boolean isEditMode() {
        return editMode;
    }

    public void setEditMode(boolean editMode) {
        this.editMode = editMode;
    }

    public void edit() {
        editMode = true;
    }

    public void save() {
        editMode = false;
    }
}
```

Facelet bean:

```
package faceletsArtikel.source;

public class AddressBookView implements
SaveEventListener {
    private AddressBookHolder addressBookHolder;

    public AddressBookHolder getAddressBookHolder() {
        return addressBookHolder;
    }

     public void setAddressBookHolder(AddressBookHolder
addressBookHolder) {
        this.addressBookHolder = addressBookHolder;
    }
}
```

1. Op het Persoon-object

Als de nieuwRecord() methode op het persoonobject geplaatst wordt, is het persoonobject als attribuut meegegeven en zou de Facelet deze methode aan kunnen roepen om een nieuw TelefoonBoekRecord-object toe te voegen.

Hoewel dit op het eerste gezicht een goede optie lijkt, houdt dit in dat het Persoon domein-object 'vervuild' wordt met viewgerelateerde methoden. Deze methoden worden als best practice op de view beans gehouden.

De andere opties zijn om de nieuwRecord() methode op de backing bean van de main view of op de backing bean van de Facelet te plaatsen. Beide opties brengen problemen met zich mee.

Het is vaak noodzakelijk dat een Facelet reageert op handelingen in de main view.

2. Op de backing bean van de main view

Als de nieuwRecord() methode op de backing bean van de main view geplaatst wordt, houdt dit in dat de main view weet heeft van het feit dat de telefoonboek-Facelet gebruikt wordt op de main view. Ook deze optie voldoet niet, aangezien het uitgangsprincipe was dat de Facelets kunnen worden toegevoegd zonder dat daarvoor de code van de pagina moet worden aangepast.

3. Op de backing bean van de Facelet

De backing bean van de Facelet is wat betreft design het beste, maar is met de huidige mogelijkheden van JSF niet mogelijk. Zoals omschreven is het niet mogelijk om via de xhtml van een pagina attributen aan de backing bean van een Facelet door te geven. In de nieuwRecord() methode hebben we een adresBoekHolder object nodig om aan dat object het nieuwe TelefoonBoekRecord toe te voegen. Een mechanisme is nodig om via de xhtml van een pagina een Java-object door te geven aan de backing bean van een Facelet.

De SetProperty tag

De SetProperty tag is geschreven om het mogelijk te maken om via de xhtml van een pagina, objecten door te geven aan een backing bean van een Facelet. De SetProperty tag overerft van het JSF UIComponentBase-object en maakt gebruik van de standaard JSF- fasesom een object door te geven. JSF kent zes fases: restore view, apply request values, process validations, update model values, invoke application en render response.

Door in de restore view fase een waarde op de target bean te zetten, heeft vanaf dat moment de target bean de doorgegeven waarde tot zijn beschikking.

Het gebruik van de SetProperty tag is zeer eenvoudig:

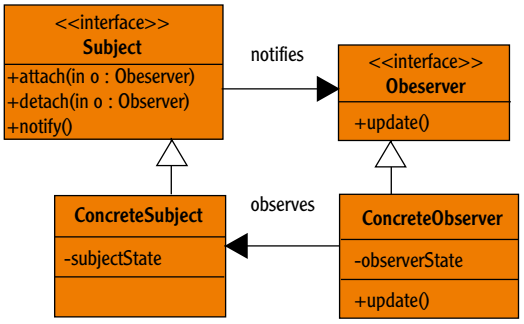
```
<example:setProperty      value="#{exampleValue}"
target="#{exampleView.target}" />
```

Door middel van de SetProperty tag is het in het voorbeeld van de vorige paragraaf nu mogelijk geworden om de nieuwRecord() methode op de backing bean van de Facelet te plaatsen. Via de SetProperty tag kan het Persoon-object worden doorgegeven. De backing bean van de Facelet beschikt daardoor over het AddressBookHolder-object. De main view backing bean heeft nu geen enkele wetenschap welke Facelets gebruikt moeten worden. De main view backing bean bevat nu alleen logica en domeinobjecten binnen zijn eigen context, logica gedelegeerd naar Facelets valt volledig buiten de main view, zowel voor de xhtml als de backing bean.

Luisteren naar events

Tot nu toe is de eenvoudige situatie besproken waarbij de Facelet onafhankelijk van de main view kan opereren. In realworldapplicaties is het vaak noodzakelijk dat een Facelet reageert op handelingen

die op de main view plaatsvinden. Daarom zijn in het voorbeeld in de main view twee mogelijke toestanden gegeven: bewerken en bekijken. Wanneer de main view in de 'bewerken'-toestand verkeert is het mogelijk om de huidige telefoonboekrecords te wijzigen of nieuwe toe te voegen. Voor het voorbeeld wordt gesteld dat de telefoonboekrecords gevalideerd moeten worden op het moment dat de gebruiker op de 'bewaren'-knop drukt.
Hier geldt ook dat de validatielogica op het Persoon-object of op de backing bean van de main view geplaatst kan worden. Maar om dezelfde redenen als hierboven genoemd, is dit niet wenselijk.
Kortom, de validatie zou op de backing bean van de Facelet geplaatst kunnen worden. Dat kan gerealiseerd worden door een combinatie van de SetProperty tag en het Observer pattern:



Door een SetProperty tag toe te voegen aan de Facelet waarmee de backing bean van de main view doorgegeven kan worden, kan de backing bean van de Facelet als Observer van de backing bean van de main view functioneren. De backing bean van de main view is in dit geval het Subject. Door dit pattern toe te passen is de Facelet dus in staat om te reageren op acties van de main view. Onderstaande code toont de code van ons voorbeeld, ditmaal met het Observer pattern toegepast.

Main view bean als Subject:

```
public class ExampleViewEnhanced implements
SaveEventListenable {
    private Persoon persoon;
    private boolean editMode;
private List<SaveEventListener> saveEventListeners = new
        ArrayList<SaveEventListener>();

    public ExampleViewEnhanced() {
        persoon = new Persoon();
    }

    ...

    public void save() {
        try {
            fireSaveEvent();
            editMode = false;
        } catch (ValidationException ve) {
          FacesContext.getCurrentInstance().addMessage(
                    null,
       new FacesMessage(FacesMessage.SEVERITY_ERROR, ve
         .getErrorMessage(), ve.getErrorMessage()));
        }
    }
private void fireSaveEvent() throws ValidationException
{
        SaveEvent event = new SaveEvent(this);
     for (SaveEventListener listener : saveEventListeners)
{
            listener.saveHappened(event);
        }
    }
public   void   addSaveEventListener(SaveEventListener
listener) {
        saveEventListeners.add(listener);
    }

    public void removeAllSaveEventListeners() {
        saveEventListeners.clear();
    }

   public void removeSaveEventListener(SaveEventListener
listener) {
        saveEventListeners.remove(listener);
    }
}

Facelet bean als Observer:
public class AddressBookView implements SaveEventListener
{
    private AddressBookHolder addressBookHolder;

    public void nieuwRecord() {
        TelefoonBoekRecord record = new
TelefoonBoekRecord();
        addressBookHolder.addTelefoonBoekRecord(record);
    }

    ...

   public void setSaveEventListenable(SaveEventListenable
        saveEventListenable) {
        saveEventListenable.addSaveEventListener(this);
    }

      public void saveHappened(SaveEvent event) throws
ValidationException {
                  for  (TelefoonBoekRecord  record  :
addressBookHolder
                .getTelefoonBoekRecords()) {
     if (record.getNaam().equals("") || record.getNaam() ==
null) {
                   ValidationException exception = new
ValidationException();
                exception
              .setErrorMessage("Het naam veld moet altijd
gevuld
                                                  zijn");
                throw exception;
            }
        }
    }
}
```

Samenvattend

Dit artikel beschrijft een design pattern voor Facelets-applicaties, waarmee de herbruikbaarheid, onderhoudbaarheid en leesbaarheid van code verbeterd wordt. De backing beans van pagina's in de applicatie dienen als Subject in het Observer-pattern uitgevoerd te worden. Hierdoor kunnen Facelets reageren op acties die op de main view plaatsvinden. Door gebruik te maken van de SetProperty tag binnen Facelets is het niet langer nodig dat de backing bean van een pagina kennis heeft van de Facelets die gebruikt worden op een pagina. Hierdoor is het in een later stadium makkelijk om een pagina uit te breiden met een Facelet zonder verder de code aan te hoeven passen en is het mogelijk om op een centrale plek de logica van de Facelet uit te breiden of aan te passen. «

Pagina's kunnen later uitgebreid worden met een Facelet zonder de code aan te passen.

Referenties

- Alle bron code voor dit voorbeeld is te downloaden op http://www.hinttech.nl/files/faceletsArtikelWithMaven.zip
- Tuning JSF Applications – Session size matters, http://www.nljug.org/pages/events/content/jspring_2008/sessions/00017/