Methodology

JSF is de techniek die sinds de J2EE specificatie versie 5 (JEE5) als standaardtechniek voor de presentatielaag wordt aanbevolen. Over het gebruik van JSF in applicaties is al veel geschreven. Over het maken van eigen componenten met behulp van JSF en het gebruik van AJAX hierbij is erg weinig geschreven. Daarom laat dit artikel een methode zien om met behulp van JSF een eigen component te maken.

JSF componenten maken met AJAX

Met een goed stappenplan simpel in gebruik

Mark Bakker
is Senior Consultant bij Xebia.

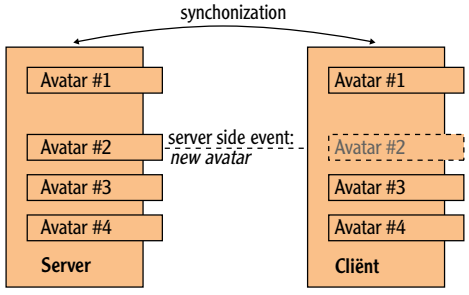
Deze methode is bedoeld om te zorgen dat je sneller componenten kunt maken en je hierbij kunt focussen op één onderdeel tegelijk. Ik laat deze methode zien aan de hand van een voorbeeld. Hiervoor wordt een component gebruikt dat is ontwikkeld voor gebruik in een chatapplicatie. Het gaat hier om een avatar component. Dit is het icoontje dat de gebruiker voorstelt in een chatwindow en dat voor de gebruiker 'praat'.

Voorbeeld van een avatar.

Dit avatar component zal asynchrone updates ondersteunen (Ajax) en kan als voorbeeld dienen om later complexere componenten te maken. De codevoorbeelden zijn niet compleet, ik heb foutafhandelingscode en andere niet specifieke code weggelaten voor de duidelijkheid.

<tussenkop>JSF in combinatie met AJAX

Om asynchrone communicatie te ondersteunen wordt er in JSF gebruik gemaakt van het dual DOM (document object model) principe. Hierbij houdt de server een DOM tree bij met daarin de staat van alle componenten op een bepaalde pagina.



Daarnaast houdt de browser een DOM tree bij met daarin de volledige staat van alle componenten op de client.

Als er een wijziging plaatsvindt op de client is de client verantwoordelijk om deze door te sturen naar de server. Op de server wordt de wijziging dan weer in de DOM tree verwerkt en worden er eventueel aanvullende acties uitgevoerd als dit nodig is. Als er aan de serverkant een wijziging plaatsvindt moet de server een asynchrone update sturen. Het JSF framework zorgt voor de synchronisatie tussen de cliënt en de server.

Er zijn twee mogelijke JSF implementaties die worden gebruikt om in JSF aan te geven welke delen van de pagina worden beïnvloed door welke componenten.

- Het per component aangeven welke componenten beïnvloed worden als er een bepaalde wijziging op het component plaatsvindt. Deze manier kost relatief veel werk om uit te programmeren. Een voorbeeld van deze techniek is ajax4jsf.
- Het gebruikte Framework zelf laten bepalen wat er allemaal geüpdate moet worden. Dit kost relatief veel server capaciteit. Als het echter goed wordt geïmplementeerd door een Framework kan dit meevallen. Een voorbeeld van deze techniek is Icefaces.

In dit artikel wordt de tweede techniek gebruikt met Icefaces als JSF implementatie.

Stappenplan

JSF componenten, en zeker JSF componenten die Ajax gebruiken zijn over het algemeen componen-

ten die bestaan uit veel losse onderdelen. Om deze onderdelen op een efficiënte manier te ontwikkelen is het van belang om een stappenplan te volgen. Dit stappenplan zorgt ervoor dat je niet teveel probeert te maken en/of aan te passen op hetzelfde moment waardoor je niet meer ziet of het probleem in de html, de css, de javascript of de java code zit. We bekijken de verschillende stappen met als voorbeeld het Avatar component.

Stap 1: De eerste schets

Voordat je een component als JSF component gaat maken is goed om eerst te weten dat het component als losstaand dhtml (html + javascript) werkt. Zo heb je een soort schets van het JSF component dat je wilt maken. Na het maken van deze schets weet je in ieder geval dat je niet hoeft te zoeken in de dhtml code als het component niet werkt. Bij het maken van de html is het handig om hierbij zoveel mogelijk de css styling los te houden van het feitelijke component. Dit zorgt ervoor dat een gebruiker van het component nog veel invloed houd op de styling.

```
<div id="j_id12:j_id22" onmousedown="grab(this);" …>
<div name="Avatar" …></div>
<div name="AvatarTEXT" …></div>
<div name="AvatarNAME" …></div>
<img src="./img/ICEfaces-logo.gif" …/>
<input id="componentId_left" onChange="iceSubmitPartial
 (form, this, event);" style="display: none;"type="text"
  value="200" />
<input id "componentId _top" onChange="iceSubmitPartial
 (form, this, event);" style="display: none;"type="text"
 value="100" />
</div>
```

De html

Zoals je kunt zien aan bovenstaand codefragment wordt er een div element gedefinieerd. Als de muisknop boven dit element wordt ingedrukt zal er een javascript functie grab worden aangeroepen. Deze grab funtie zorgt ervoor dat het component in de verplaatsbare mode komt en je het dus rond kunt slepen over je scherm.

```
function grab(context){
    document.onmousedown = falsefunc; // in NS this
 prevents cascading of events, thus disabling text
 selection
    dragobj = context;
    document.onmousemove = drag;
    document.onmouseup = drop;
    grabx = mousex;
    graby = mousey;
    elex = orix = dragobj.offsetLeft;
    eley = oriy = dragobj.offsetTop;
    update();
}
```

De grab functie

In bovenstaande javascript functie zie je dat er op de events onmousemove en onmouseup de functies drag en drop worden gezet. Met behulp van drag wordt het component verplaatst en eens per seconde wordt de nieuwe positie naar de server verstuurd.De drop functie zorgt ervoor dat je weer terug gaat naar de begin staat en een laatste update naar de server stuurt. Het component is hierna dus pas weer verplaatsbaar door er met de muis op te klikken. De drag functie is als volgt uitgewerkt:

```
function drag(e){ // parameter passing is important for
 NS family
  if (dragobj){
    elex = orix + (mousex-grabx);
    eley = oriy + (mousey-graby);

    dragobj.style.position = "absolute";
    dragobj.style.left = (elex).toString(10) + 'px';
    dragobj.style.top = (eley).toString(10) + 'px';
    document.getElementById(dragobj.id+"_left").value =
 (elex).toString(10);
    document.getElementById(dragobj.id+"_top").value =
 (eley).toString(10);

    if(!dispatchPosition){
        setTimeout('dispatchChangeEvent("'+dragobj.id+'_
left")',1000);
        setTimeout('dispatchChangeEvent("'+dragobj.id+'_
top")',1000);
        setTimeout('dispatchPosition=false;',1000);
        dispatchPosition=true;
    }
  }
  update(e);
  return false; // in IE this prevents cascading of
 events, thus text selection is disabled
}
```

De drag functie

Zoals je ziet wordt de functie dispatchChangeEvent aangeroepen. Deze functie zorgt ervoor dat de onChange functie op het hidden field [coponentId]_left wordt aangeroepen. Op de onChange staat in de html dat er een synchronisatie met de server plaats moet vinden als de input van het veld verandert. Hiermee heb je de belangrijkste code gezien die je nodig hebt aan de clientzijde en is de schets voor het avatar component dus compleet. De volgende stap is het maken van het JSF component.

Stap 2: Het component

Een component bestaat uit twee delen, te weten een UI class met al zijn attributen en een renderer. De UI class zelf is meestal een extensie van een UIInput basis component als er data vanaf de client komt (je hebt ook componenten die alleen output tonen). Hieronder zie je een voorbeeld van het Avatar component.

```
public class Avatar extends UIInput {
    public static final String COMPONENT_FAMILY =
 "mark. chat";
    public static final String COMPONENET_TYPE =
"com.mark.avatar.Avatar";
    public static final String DEFAULT_RENDERER_TYPE =
 "com.mark.avatar.AvatarRenderer";
    private static final Resource AVATAR_JS =
new JarResource("com/mark/avatar/avatar.js");

    private Integer top;
    private Integer left;
    private String message;
    private String name;
    private String image;
    private URI baseURI;

    public String getFamily(){
        return COMPONENT_FAMILY;
    }

    public String getRendererType() {
```

Maak een stappenplan om te voorkomen dat je teveel maakt en het overzicht kwijt raakt.

De renderer is het meest interessante deel van de JSF component.

```
        return DEFAULT_RENDERER_TYPE;
    }

    public String getComponentType() {
        return COMPONENET_TYPE;
    }
…
    public void setTop(Integer top) {
         System.err.println("Setting top to: "+top);
        this.top = top;
    }

    public Integer getTop() {
        if (top != null) {
            return top;
        }
        ValueBinding vb = getValueBinding("top");
        return (vb != null ? (Integer)
vb.getValue(getFacesContext()) : 0);
    }
…
more setters and getters
```

Het interessantste deel van een JSF component is de renderer. Hierin wordt een encode en decode gedaan. Dit is het wegschrijven van updates naar de client, deze worden later aan de clientkant in de HTML DOM tree gesynchroniseerd. Dit synchroniseren wordt gedaan door Icefaces. Het verwerken van updates vanaf de client gebeurt in de decode functie.
Hieronder zie je de AvatarRenderer.

```
public class AvatarRenderer extends
 DomBasicInputRenderer{
    public AvatarRenderer() {
        super();
    }

    public void decode(FacesContext facesContext,
 UIComponent uiComponent) {
        validateParameters(facesContext, uiComponent,
 null);
        Avatar uiChat = (Avatar)uiComponent;
        String clientId = uiComponent.
getClientId(facesContext);
        if (clientId == null) {

System.out.println("Client id is not defined for
decoding");
        }


Map requestMap = facesContext.getExternalContext().
getRequestParameterMap();
        String leftString = (String)requestMap.
get(clientId +
"_left");
        String topString = (String)requestMap.
get(clientId +
"_top");
        …
        Integer left = Integer.parseInt(leftString);
        uiComponent.broadcast(new ValueChangeEvent(uiCom
ponent,uiChat.getLeft(),left));
        uiChat.setLeft(left);
        Integer top = Integer.parseInt(topString);
        uiComponent.broadcast(new ValueChangeEvent(uiCom
ponent,uiChat.getTop(),top));
        uiChat.setTop(top);
    }

    public void encodeEnd(FacesContext facesContext,
 UIComponent uiComponent) throws IOException {
            Avatar avatar = (Avatar)uiComponent;
            DOMContext domContext = DOMContext.
attachDOMContext(facesContext, uiComponent);
            String clientId = avatar.
getClientId(facesContext);

            …
```

```
            if (!domContext.isInitialized()) {
                root = domContext.
createRootElement("div");
                setRootElementId(facesContext, root,
 uiComponent);

                root.setAttribute("onmousedown","grab
(this);");

                Element avatarElement = domContext.
createElement("div");

                if(avatar.isMoveable()){

avatarElement.setAttribute("style","position: relative;
 width: 50; height: 70;");
                }else{

avatarElement.setAttribute("style","position: relative;
 width: 50; height: 70;");
                }

                Element leftHiddenInput = domContext.
createElement("input");

leftHiddenInput.setAttribute("id",uiComponent.
getClientId(facesContext)+"_left");
                leftHiddenInput.
setAttribute("type","text");

leftHiddenInput.setAttribute("name",uiComponent.
getClientId(facesContext)+"_left");

leftHiddenInput.setAttribute("value",""+avatar.
getLeft());

leftHiddenInput.setAttribute("onChange",this.
ICESUBMITPARTIAL);
                root.appendChild(leftHiddenInput);
                …
                the right hidden field
            //base position

root.setAttribute("style","position: absolute; left:
 "+avatar.getLeft()+"px; top: "+avatar.getTop()+"px;");

            //image
            avatarImage.setAttribute("src","."+avatar.
getImage());

avatarImage.setAttribute("style","position: relative;
 top: -135; ");

            …
    }

}
```

In de decode functie wordt de input van de client vertaald naar ValueChangeEvents. Hier kan een gebruiker van het component weer naar luisteren binnen zijn applicatie.
Daarnaast zie je dat hier op de hidden fields een onChange functie wordt gezet met als inhoud this.ICESUBMITPARTIAL deze functie zorgt dat de hidden fields worden verstuurd en in de decode functie verwerkt kunnen worden.
Na het maken van van deze twee klassen is het nog een kwestie van het maken van een paar configuratiefiles om het geheel te laten werken. Deze configuratie wordt behandeld in de volgende stap.

Stap 3: De facelet configuratie

Na het maken van het component zelf is het nog nodig om het component te registreren in de facelet configuratie. Hieronder staat de avatar facelet configuratie. Deze moet in de META-INF/facelet directory gezet worden om opgepakt te worden Het

bestand heet avatar.taglib.xml.

```
<?xml version="1.0"?>
<!DOCTYPE facelet-taglib PUBLIC
  "-//Sun Microsystems, Inc.//DTD Facelet Taglib 1.0//
EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<facelet-taglib>
        <namespace>http://chat.nl/jsf/component/tags
</namespace>
        <tag>
                <tag-name>avatar</tag-name>
                <component>
                        <component-type>com.mark.
avatar.Avatar</component-type>
                        <renderer-type>com.mark.
avatar.AvatarRenderer</renderer-type>
                        <handler-class>com.
icesoft.faces.component.facelets.IceComponentHandler
</handler-class>
                </component>
        </tag>


</facelet-taglib>
```

Hier wordt een koppeling gemaakt tussen het component en redererer type zoals dat is gedefinieert in het component. Deze twee attributen worden gekoppeld aan een namespace die je kunt gebruiken binnen een facelet pagina.

Ook is het nodig om een META-INF/faces-config.xml bestand te maken met de volgende inhoud:

```
<?xml version="1.0"?>

<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config
 1.1//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
        <render-kit>
          <description>
                Custom Components
          </description>
                <!-- Custom components -->
           <renderer>
                <component-family>mark.chat
</component-family>
                <renderer-type>com.mark.avatar.
AvatarRenderer</renderer-type>
                <renderer-class>com.mark.avatar.
AvatarRenderer</renderer-class>
           </renderer>
        </render-kit>

        <component>
           <component-type>com.mark.avatar.Avatar
</component-type>
           <component-class>com.mark.avatar.Avatar
</component-class>
         </component>

</faces-config>
```

Dit bestand zorgt voor de registratie van de renderer bij het component.

Stap 4: Het maken van de jar file

Om het component te gebruiken is het nodig alle files in een jar file te zetten. Deze jar file kun je later binnen je chatapplicatie gebruiken. In een jar file kun je meerdere componenten stoppen. In dit geval staat alleen het Avatar component in de file.
De jar file zal er dan uitzien als avatar.jar:

```
/META-INF/faces-config.xml
                  /facelet/avatar.taglib.xml
com/mark/avatar/Avatar.class
                            AvatarRenderer.class
                            avatar.js
```

Stap 5: Gebruik het component in je applicatie

Om het component binnen je applicatie te kunnen gebruiken moet je twee dingen doen:

- De jar file toevoegen aan het classpath (bijvoorbeeld de lib directy in een war file)
- Het gebruiken van het component op een pagina

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.
dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
…
         xmlns:chat="http://chat.nl/jsf/component/tags"
         xmlns:ice="http://www.icesoft.com/icefaces/
          component">
…
<ice:dataTable value="#{chatBox.otherAvatars}"
 var="avatar">
         <ice:column>
                  <chat:avatar left="#{avatar.left}"
 top="#{avatar.top}"
                           message="#{avatar.
message}" name="#{avatar.name}"
                           image="#{avatar.image}"
 moveable="false" />
         </ice:column>
</ice:dataTable>
<chat:avatar binding="#{myAvatar}" image="/img/me.gif

         moveable="true" />
…
```

In bovenstaand codevoorbeeld zie je dat de namespace chat wordt gekoppeld aan de in het component gedefinieerde namespace http://chat.nl/jsf/component/tags Dit zorgt ervoor dat je verderop in het fragment de chat namespace kunt gebruiken.
Je ziet hierna een dataTable waarin een lijst met avirtars wordt gekoppeld aan de lokale namespace avatar.
Hierna wordt er een kolom gedefinieerd. Deze kolom wordt gebruikt om een herhaald element te kunnen gebruiken. Hierbinnen wordt het avatar component gebruikt. Deze versie van het component heeft moveable=false staan. Dit zijn de andere gebruikers van een chatbox.
Onderaan vind je een avatar component dat de aanroeper van de pagina voorstelt. Deze is verplaatsbaar. Aan de serverkant worden er met behulp van push technologie server geïnitieerde updates naar de browsers gestuurd.

Conclusie

Als je dit stappenplan volgt moet het voor veel mensen mogelijk zijn om bestaande dhtml componenten om te zetten naar JSF componenten met AJAX ondersteuning.Ik nodig hierbij dus iedereen uit om de hoeveelheid nuttige en interessante componenten die nu al beschikbaar zijn uit te breiden met nog veel meer componenten. «

Met een goed stappenplan is het eenvoudig om dtml componenten om te zetten naar JSF componenten met AJAX.

23 NOVEMBER 2010 • HOLIDAY INN LEIDEN

RICK VAN DER LANS OVER

HET ONTWERPEN VAN SERVICE ORIENTED ARCHITECTURES

RICHTLIJNEN VOOR EN ERVARINGEN MET SOA'S

Onder auspiciën van Release. Abonnees krijgen korting.

DEELNEMERS WAARDEERDEN DE VORIGE EDITIE MET EEN 8!

Dit seminar is een vervolg op het succesvolle seminar Service Oriented Architectures. Ook als u niet eerder aan Service Oriented Architectures heeft deelgenomen, is het seminar Het ontwerpen van Service Oriented Architectures uitstekend te volgen.

23 NOVEMBER 2010

HET ONTWERPEN VAN SERVICE ORIENTED ARCHITECTURES

RICHTLIJNEN VOOR EN ERVARINGEN MET SOA'S

- RICHTLIJNEN EN TECHNIEKEN VOOR HET ONTWERPEN VAN SOA'S
- ONTWERPREGELS VOOR BASIC, COMPOSITE EN BUSINESS PROCESS SERVICES
- VAN STAND-ALONE NAAR LOOSELY COUPLED, WEBSERVICES-GEBASEERDE EN GEÏNTEGREERDE SYSTEMEN

LOCATIE	Holiday Inn Leiden
TIJD	Van 9.30 uur tot 17.00 uur
REGISTRATIE	www.arrayseminars.nl

BESTEMD VOOR Ú

Het seminar is zeer geschikt voor ondermeer IT-managers, technology planners, infrastructuur-architecten, consultants, systeemanalisten en -ontwerpers, databaseontwerpers en -beheerders.

Kijk snel op www.arrayseminars.nl voor het complete programma

De meeste standaarden zijn klaar en de producten zijn beschikbaar. Maar waar begint men? Hoe dient een Service Oriented Architecture (SOA) ontworpen te worden? Welke ontwerprichtlijnen bestaan er? Wat zijn de do's en dont's voor deze baanbrekende technologie waarmee informatie-systemen geïntegreerd kunnen worden? Dit ééndaagse seminar behandelt deze cruciale richtlijnen.
Veel aandacht zal worden geschonken aan de Enterprise Service Bus (ESB). Dé technologie voor het ontwikkelen van een SOA. Het is een moderne implementatie van een SOA.
Omdat reeds diverse organisaties ervaringen hebben opgedaan met de bouw van SOA's, beginnen langzaamaan de ontwerprichtlijnen boven tafel te komen. Dit seminar is geen theoretische verhandeling en ook geen toelichting van wat een SOA is, maar het is een samenvatting van deze ervaringen. Onmisbaar voor diegenen die met een SOA gestart zijn of die overwegen een SOA te ontwikkelen.

Array Seminars: specialist in IT-vakkennis!