Glassfish, JSF, Facelets och Ajax4Jsf

Introduktion

Följande guide går snabbt igenom vad Glassfish, JSF, Facelets och Ajax4Jsf är samt beskriver enkelt hur teknikerna kan användas och hur man sätter upp applikationsservern Glassfish på Windows. Beskrivningen är inte djup och genomgående, och är inte alls att betrakta som något läromedel för teknikerna/applikationsservern. Däremot bifogas en länksamling för den intresserade och kunskapstörstande läsaren.

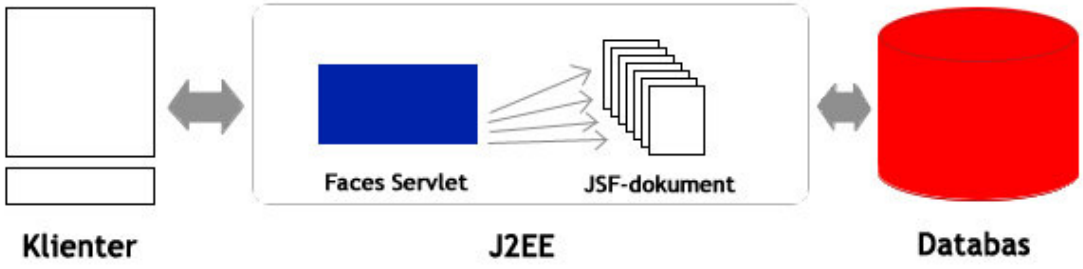
Förutom de teknologier som gås igenom, krävs en viss erfarenhet av programmering med Java och visst kunnande inom Java EE-arkitekturen som t.ex. JSP, Servlets och Java-bönor.

Kort om Glassfish

Glassfish är en gratis, öppen mjukvara (open source) från Sun och stödjer den senaste J2EE-plattformen fullt ut. Glassfish är en applikationsserver och webserver i ett, det krävs inga andra mjukvaror för att köra en webapplikation. J2EE (Java 2 Enterprise Edition) tillhandahåller teknologier som Java Servlets, Java Server Pages (JSP), Java Server Faces (JSF), Java Messaging System (JMS), Java API for Web Services (JAX-WS), Enterprise Java Beans 3.0 (EJB3), bland mycket annat. Känner du inte igen dig bland dessa teknologier bör du rimligen läsa in dig kort på J2EE innan du fortsätter med denna guide. Visst kunnande inom JSP och Servlets samt viss kännedom om Java-bönor är en förutsättning för att förstå guiden fullt ut.

Kort om JSF

Java Server Faces, från och med nu kallat JSF, är en teknologi för att presentera information till användare, främst via Internet – men även andra medier. I den här guiden riktar vi enbart in oss på helt vanlig webserver till dator-presentation. JSF förenklar utvecklingen av webapplikationer genom att tillhandahålla ett komponentcentraliserat förhållande av utvecklingen. MVC (Model-View-Controller) är en av grundstenarna i JSF.



Kort om Facelets

Facelets är en påbyggnad på JSF, och hanteras av en annan s.k. ViewHandler – dvs. den del av JSF som behandlar och renderar JSF- eller Facelets-dokumenten innan de visas för klienten. Med Facelets har man gått ett steg längre än man gjorde med JSF, man separerar helt på logik och presentation i stort sett. Med Facelets behöver du inte skriva någon Java-specifik kod i dina JSF-filer – utan du skriver XHTML rakt igenom. Man styr utseende och uppförande genom attributen på XHTML-taggar.

Kort om Ajax4Jsf

Ajax4Jsf är ett tredjepartsbibliotek för att på ett smidigt sätt använda Ajax i sina webdokument. Nu mera är Ajax4Jsf en del av Rich Faces från JBoss, men för enkelhetens skull kommer det att refereras till som Ajax4Jsf. Med Ajax4Jsf slipper du skriva JavaScript och manuellt styra dina komponenter och datainnehåll. Du definierar enbart kontrollerna i JSF och hanterar informationen fram och tillbaka från Java-bönor.

Först ut: Java-kompilator och nedladdning av paket

Innan vi sätter igång med installationer och genomgångar av de olika teknikerna, krävs det självklart att man har en Java-kompilator installerad på sitt system. För den här guiden används den senaste vid skrivandets stund, vilken är Java 6 SE. Antas görs också att miljövariabler är korrekt satta - JAVA_HOME bör peka på din installation av Java SDK och PATH bör peka mot bin-katalogen av Java SDK.

De mjukvaror och bibliotek som används för guiden är:

Glassfish V2 från https://glassfish.dev.java.net/
Facelets 1.1.14 från https://facelets.dev.java.net/
Rich Faces UI 3.2.0 från http://www.jboss.org/jbossrichfaces/

För att Facelets och Ajax4Jsf ska fungera krävs det ytterligare paket, nämligen fyra jar-filer från Apache Commons. Det är commons-beanutils.jar, commons-collections.jar, commons-digester.jar samt commons-logging.jar – dessa finns att ladda ner från http://commons.apache.org.

Installera Glassfish i Windows

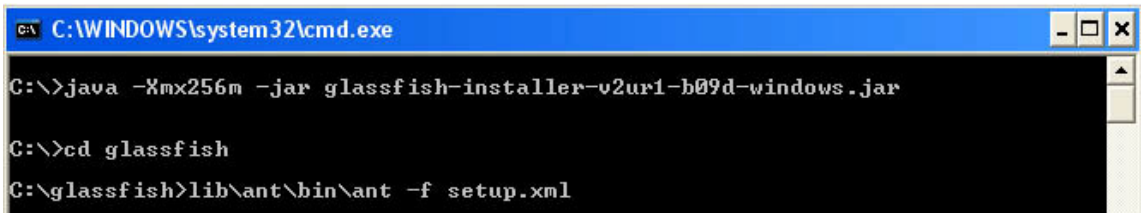
Filen som laddades ner från Glassfish websida antar vi placerades i roten på C:. Öppna kommandoprompten och stega dig ner till roten av C: och kör följande kommando för att installera Glassfish:

```
java -Xmx256m -jar glassfish-installer-v2ur1-b09d-windows.jar
```

Notera att du kan vara tvungen att ändra på filnamnet till det som din nedladdade version heter. När kommandot utförts är inte installationen helt klar, vi måste även bygga upp och konfigurera Glassfish med Ant som kom i samma paket som Glassfish. Vi bygger filen som sagt med Ant och instruktionsfilen setup.xml. I setup.xml lagras alla standardvärden som Glassfish konfigureras med, som portar och inloggningsuppgifter för administratörer. Om du vill ändra några inställningar måste du göra det innan vi går vidare. Standardport för HTTP-trafik är t.ex. 8080, standardport för administrationspanelen är 4848, standardanvändarnamnet är 'admin' och lösenord 'adminadmin'.

Då så, gå in i C:\glassfish\ och kör:

```
lib\ant\bin\ant -f setup.xml
```

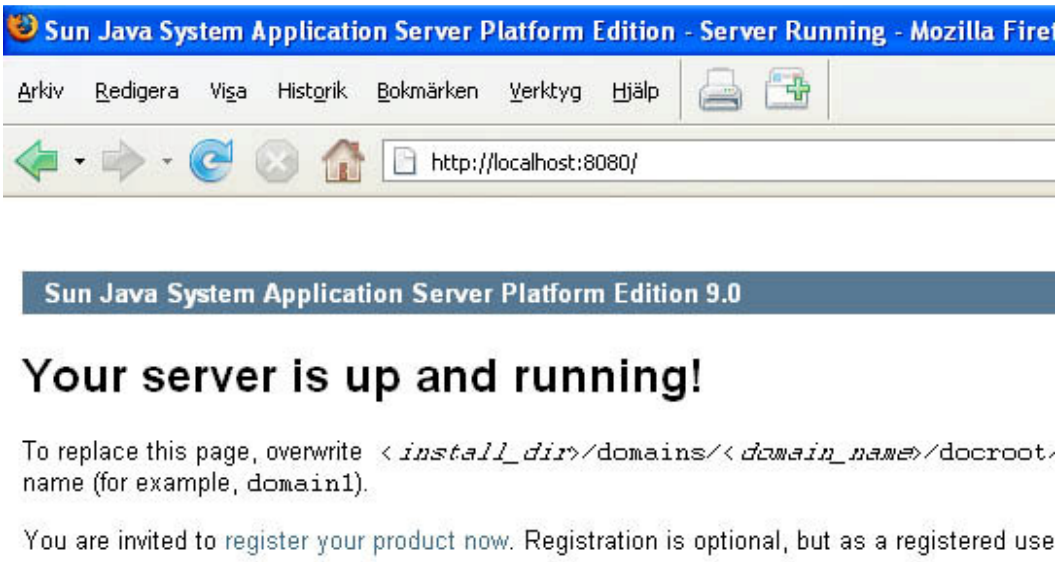


Nu är Glassfish installerat. Då ska vi bara se så att allt fungerar som det ska! Som standard anropas som sagt webservern på port 8080 (HTTP-trafik) och administrationsgränssnittet på port 4848. Det kommer även med en s.k. domän på köpet, och den kommer vi att använda genom hela guiden. För att enkelt nå alla verktyg med Glassfish från kommandoprompten rekommenderas att lägga till bin-

katalogen i din miljövariabel PATH. När du gjort det kan vi enkelt starta webservern och vår domän genom att utföra:

```
asadmin start-domain domain1
```

domain1 är namnet på domänen som skapades vid installationen. Det tar några sekunder för att få igång alla tjänster för Glassfish. När webservern startats, peka din webläsare till adressen http://localhost:8080/ och du bör få upp en sida som ser ut så här:



Som alla webservrar har Glassfish en dokumentrot, där ovanstående index-fil finns. Som standard är sökvägen till dokumentroten %GLASSFISH%/domains/domain1/docroot/ - den kan ändras innan man bygger upp sitt Glassfish med Ant. Det är här i du kan placera websidor. Vi kommer däremot inte att manuellt lägga upp någonting i dokumentroten – utan ladda upp webarkiv med applikationer via administrationsgränssnittet.

Peka om din webläsare till adressen http://localhost:4848/, och du bör få upp en inloggningssida för administrationsgränssnittet till Glassfish:

Användarnamnet är per-default satt till ’admin’ och lösenordet till ’adminadmin’. I administrationsgränssnittet kan man påverka det mesta för Glassfish-servern – från att utplacera (deploy) projekt, skapa databaspooler, sätta upp webservices till att sätta upp ett gränssnitt för JMS. Under nästa rubrik ska vi se hur man utplacerar projekt.

Första projektet: Java Server Faces

Vi ska skapa ett mycket enkelt formulär som postar data till en böna, och sedan presenterar den informationen på en efterföljande sida. Sammanlagt kommer projektet att bestå av fem källfiler – en webapplikations-beskrivning, en konfigurationsfil för JSF, en källkodsfil för vår Java-böna samt två JSF-dokument. Vi börjar i botten med vår Java-böna som håller informationen åt oss. Det är absolut inget märkvärdigt med den här bönan, det är ett enkelt POJO (Plain Old Java Object).

```
package guide.jsf;

public class JsfBean {
 private String firstName;
 private String lastName;

 public String getFirstName() {
     return this.firstName;
 }
 public void setFirstName(String firstName) {
     this.firstName = firstName;
 }
 public String getLastName() {
     return this.lastName;
 }
 public void setLastName(String lastName) {
     this.lastName = lastName;
 }
}
```

En viktig detalj är att getter- och settermetoder måste finnas för de klassvariabler som ska användas. Utan vidare förklaringar över filinnehållet: Döp filen till `JsfBean.java` och placera den i en mapp döpt till `jsfprojekt/src/` och kompilera den. Flytta sedan den genererade `JsfBean.class` till mappen `jsfprojekt/WEB-INF/classes/guide/jsf/` – dessa mappar måste du skapa först såklart. Senare kommer vi att se en överblick över filstrukturen för hela projektet.

Vi fortsätter med att skapa våra JSF-dokument. Som alla JSP-sidor kräver även JSF instruktioner om hur det ska behandla innehållet i filen. Vi börjar med att deklarera vad det är för typ av sida, samt att tala om för JSF vilka taggar och prefix som ska användas.

```
<%@ page language="Java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
```

När vi sedan ska markera upp taggar i vårat dokument som ska behandlas av JSF använder vi alltså taggarna ’f’ för JSF-specifika taggar och ’h’ för HTML-taggar som ska renderas genom JSF. Hela dokumentet ser ut som följer:

```
<%@ page language="Java" contentType="text/html; charset=UTF-8"
                pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=UTF-8">
<title>Enter Data</title>
</head>
<body>

<f:view>
 <h:form>
 <h:outputText value="First Name:"/>
 <h:inputText label="First Name"
                value="#{JsfBean.firstName}">
 </h:inputText>

 <h:outputText value="Last Name:"/>
 <h:inputText label="Last Name"
                value="#{JsfBean.lastName}">
 </h:inputText>

 <h:commandButton action="save"
                value="Save"></h:commandButton>
 </h:form>
</f:view>

</body>
</html>
```

Spara filen som `save_data.jsp` och placera den i rotkatalogen `jsfprojekt`. Som du märker är formuläret inte försett med något action-attribut, det sitter däremot på vår commandButton. Våra konfigurationsfiler sköter navigeringen mellan våra sidor, som vi snart ska se. Vi visar direkt sidan som presenterar informationen:

```
<%@ page language="Java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=UTF-8">
<title>Saved Data</title>
</head>
<body>

<f:view>
 <h:panelGrid columns="2">
 <h:outputText value="First Name:" />
 <h:outputText value="#{JsfBean.firstName}" />
 <h:outputText value="Last Name:" />
 <h:outputText value="#{JsfBean.lastName}" />
 </h:panelGrid>
</f:view>

</body>
</html>
```

Sidan förklarar sig själv – vi använder JSF-taggar för JSF's **ViewHandler** att behandla, där vår data ska presenteras. All information inom view-elementet för JSF som inte är JSF-specifik kommer att avfärdas och därför inte heller att visas. Döp filen till `present_data.jsp` och placera den i samma mapp som förut, delvis roten i `jsfprojekt`. Nu kommer vi till själva kärnan som får vårat projekt att behandlas som JSF, nämligen de två konfigurationsfiler som krävs. Den ena är specifik för JSF, den andra är välkänd om du har arbetat med Java EE tidigare – nämligen `web.xml`. Vi börjar med den senare:

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
             http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
   <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
   </servlet-mapping>
</web-app>
```

Vi börjar med att deklarera en Servlet – nämligen den JSF-specifika Servlet som JSF använder. För att Glassfish sen ska känna igen vilka dokument som ska behandlas som JSF-dokument lägger vi till en

mappning för JSF-dokumenten med filnamnet jsf. Notera att vi trots detta använder JSP-dokument i grunden, med filtillägget *.jsp. För att våra JSF-filer ska behandlas genom JSF krävs det att vi anropar filerna med tiltillägget *.jsf – annars behandlas de inte som JSF-dokument utan som rena JSP-dokument. Spara filen som `web.xml` i mappen `jsfprojekt/WEB-INF/`. För att vår Java-böna ska fungera samt navigeringen mellan våra två sidor, vi använde ju oss inte av action-attributet på vårat formulär, krävs en JSF-specifik konfigurationsfil – nämligen `faces-config.xml`. Den ser ut som följer:

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>
    <managed-bean>
     <managed-bean-name>JsfBean</managed-bean-name>
     <managed-bean-class>guide.jsf.JsfBean</managed-bean-class>
     <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>

    <navigation-rule>
     <from-view-id>/save_data.jsp</from-view-id>
     <navigation-case>
     <from-outcome>save</from-outcome>
     <to-view-id>/present_data.jsp</to-view-id>
     </navigation-case>
    </navigation-rule>
</faces-config>
```

Vi deklarerar att vi vill använda en Java-böna i projektet genom elementet managed-bean och anger sedan vad bönan ska heta, var den ligger och livstiden för bönan. Eftersom att vi enbart ska skicka data mellan två filer anger vi `request` som livstid. Det finns ytterligare två alternativ, `session` eller `scope`. Det är allt som krävs för att registrera vår böna för projektet.

Vi behöver också tala om hur navigeringen sker från vår fil save_data.jsp till present_data.jsp. Här är nyckelelementet `from-outcome`. I save_data.jsp satte vi attributet `action` med värdet ”save” på vår commandButton – det är det vi registrerar här. Om knappen trycks på, postas formuläret och Glassfish vet då var navigeringen ska ske efter att bönan har registrerat värdena – nämligen present_data.jsp. Nu har vi alla våra filer klara, och de bör vara placerade i en filstruktur som ser ut så här:

```
jsfprojekt/
      WEB-INF/
           classes/
                se/
                    guider/
                         jsf/
                              JsfBean.class
           faces-config.xml
           web.xml
      src/
           JsfBean.java
      save_data.jsp
      present_data.jsp
```

Nu är det dags att paketera vårat projekt för att sedan kunna utplacera (deploy) det till vår Glassfish-server. Vi ska paketera projektet till ett webarkiv, s.k. WAR-arkiv – ett Web ARchive. Ett WAR-arkiv är inget annat än en jar-fil egentligen – så vi paketerar självklart med Javas program `jar`. Öppna kommandoprompten och placera dig i katalogen jsfprojekt. Kör sedan följande kommando:

```
jar –cvf jsfprojekt.war *
```

Nu kommer alla filer i katalogen att paketeras i filen `jsfprojekt.war` och vi kan sedan enkelt utplacera den till vår server. Se till att du har servern igång (`asadmin start-domain domain1`) och peka sedan din webläsare till http://localhost:4848 och logga in i administrationsgränssnittet för Glassfish. I menyn till vänster, gå in på Applications/Web Applications och klicka på knappen ”Deploy...”. Klicka på ”Bläddra” och leta upp det WAR-arkiv som vi nyss skapade och klicka vidare på ”Next”. Skärmen efter laddar in default-värden för projektet, som t.ex. URL och applikationsnamn. Vi behöver inte ändra på något, så klicka på ”Finish” för att slutföra utplaceringen. Förhoppningsvis visas inga felmeddelanden och du kommer tillbaka till sidan för Web Applications – den borde nu se ut så här:

Peka sedan till webläsare till http://localhost:8080/jsfprojekt/save_data.jsf - notera att filtillägget ska vara jsf – inte jsp. Förhoppningsvis får du nu upp ett fungerande första JSF-projekt!

Andra projektet: Utveckla projekt 1 med Facelets

Vi ska nu skapa ett liknande projekt med Facelets, med i stort sett samma katalogstruktur och filer. Eftersom att Facelets använder en annan ViewHandler och arbetar utifrån XHTML-dokument (när det gäller web) så är det några förändringar som måste ske. Vi börjar med `save_data.xhtml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!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:h=http://java.sun.com/jsf/html
    xmlns:f="http://java.sun.com/jsf/core">
<head>
<meta http-equiv="Content-Type"
    content="text/html; charset=UTF-8" />
<title>Enter Data</title>
</head>
<body>
<form jsfc="h:form" id="nameForm">
<label jsfc="h:outputLabel" for="firstNameField">Firstname:</label>
<input type="text" jsfc="h:inputText"
    value="#{FaceletBean.firstName}" id="firstNameField" />
<br />
<label jsfc="h:outputLabel" for="lastNameField">Lastname:</label>
<input type="text" jsfc="h:inputText"
    value="#{FaceletBean.lastName}" id="lastNameField" />
<br />
<h:selectOneRadio id="selectSex" value="#{FaceletBean.sex}">
    <input type="radio" jsfc="f:selectItem" itemValue="Male"
        id="maleBtn" itemLabel="Male" />
    <label jsfc="h:outputLabel" for="maleBtn"
        rendered="false">Male</label>
    <input type="radio" jsfc="f:selectItem" itemValue="Female"
        id="femaleBtn" itemLabel="Female" />
    <label jsfc="h:outputLabel" for="femaleBtn"
        rendered="false">Female</label>
</h:selectOneRadio>
<br />
<span jsfc="h:outputText">Interests:</span><br />
<h:selectManyCheckbox value="#{FaceletBean.interests}">
    <input type="checkbox" jsfc="f:selectItem"
        itemValue="Sport" id="sportBox" itemLabel="Sport" />
    <label jsfc="h:outputLabel" for="sportBox"
        rendered="false">Sport</label>
    <input type="checkbox" jsfc="f:selectItem"
        itemValue="Food" id="foodBox" itemLabel="Food" />
    <label jsfc="h:outputLabel" for="foodBox"
        rendered="false">Food</label>
    <input type="checkbox" jsfc="f:selectItem"
        itemValue="Java EE" id="javaeeBox" itemLabel="Java EE" />
    <label jsfc="h:outputLabel" for="javaeeBox"
        rendered="false">Java EE</label>
</h:selectManyCheckbox><br />
<input type="submit" jsfc="h:commandButton" value="Send"
    action="submit" id="submitBtn" />
</form></body></html>
```

Som synes används XHTML-taggar vid presentation vid element. Vissa JSF-taggar kommer man inte ifrån, som när man definierar typen av elementen. För att definiera ett element som JSF-element med Facelets används attributet `jsfc` följt av motsvarigheten/typen i JSF. En span-tagg t.ex. motsvaras i JSF av `h:outputText` för att visa text och information. På checkboxar och radioknappar används ett attribut som heter `itemLabel`. Det fungerar som (X)HTMLs `label`-tagg, och visar relaterad rubrik för knappen/checkboxen. Det kan se lite förvirrat ut att deklarera både en itemLabel och ett label-element – men det finns en tanke bakom. Om du skulle öppna den här filen i valfri webläsare, utan att JSF/Facelets renderar sidan, så skulle sidan se ut exakt som den kommer att göra **efter** att den har renderats. I och med att vi sätter attributet `render="false"` på label-elementen kommer dessa att plockas bort efter att Facelets har gjort sitt med renderingen och istället använda attributet `itemLabel` från `input`-taggen.

Eftersom att det här är ett XHTML-dokument och inte ett JSF-dokument, sätter vi inte `page`-attribut eller `taglib`-attribut som vi gjorde i föregående projekt. XHTML-dokumenten har ingen Java-specifik kod i sig, enbart JSF- och Facelets-taggar. Därför ser början av dokumentet annorlunda ut än om vi skulle ha använt enbart JSF. Dock är det viktigt att sätta upp rätt namnrymder (eng. namespaces) för Facelets och JSF i `html`-elementet. Spara filen som `save_data.xhtml` (notera att det är ett XHTML-dokument – och **inte** ett JSF/JSP-dokument! Därav .xhtml som filändelse) i rotkatalogen för vårat nya projekt – `faceletsprojekt`.

Vi fortsätter med filen `present_data.xhtml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!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:h="http://java.sun.com/jsf/html">
<head>
<meta http-equiv="Content-Type"
    content="text/html; charset=UTF-8" />
<title>Saved Data</title>
</head>
<body>

<b>Firstname:</b><br />
<span jsfc="h:outputText" value="#{FaceletBean.firstName}" /><br />
<b>Lastname:</b><br />
<span jsfc="h:outputText" value="#{FaceletBean.lastName}" /><br />
<b>Sex:</b><br />
<span jsfc="h:outputText" value="#{FaceletBean.sex}" /><br />
<b>Interests:</b><br />
<span jsfc="h:outputText" value="#{FaceletBean.interestsList}" />
<br />

</body>
</html>
```

Samma sak i den här filen, ingen Java-specifik kod – enbart Facelets-taggar och EL-taggar. Spara filen som `present_data.xhtml` i rotkatalogen (`faceletsprojekt`).

Nu kommer vi till vår Java-böna – som du kanske hade gissat kallar vi den för FaceletBean i det här projektet. Det är, precis som i föregående projekt, en simpel POJO – Plain Old Java Object.

```
package guide.facelets;

public class FaceletBean {
    private String firstName;
    private String lastName;
    private String sex;
    private String[] interests;

    public String getFirstName() {
        return this.firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return this.lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public String getSex() {
        return this.sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
    public String[] getInterests() {
        return this.interests;
    }
    public void setInterests(String[] interests) {
        this.interests = interests;
    }
    public String getInterestsList() {
        if(this.interests == null)
            return "";

        StringBuffer stringBuffer = new StringBuffer();

        for(String interest : this.interests) {
            if(!stringBuffer.toString().equals(""))
                stringBuffer.append(", ");

            stringBuffer.append(interest);
        }

        return stringBuffer.toString();
    }
}
```

Förhoppningsvis är bönan självförklarande. Spara den som `faceletsprojekt/src/FaceletBean.java`, kompilera den och placera `FaceletBean.class` i katalogen `faceletsprojekt/WEB-INF/classes/guide/facelets/`. Vi fortsätter med web.xml, och här måste det till lite förändringar för att Facelets ska ta över som ViewHandler efter JSF:

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<display-name>Facelet Example</display-name>
    <context-param>
        <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
        <param-value>.xhtml</param-value>
    </context-param>
    <context-param>
        <param-name>facelets.DEVELOPMENT</param-name>
        <param-value>true</param-value>
    </context-param>
    <context-param>
        <param-name>com.sun.faces.validateXml</param-name>
        <param-value>true</param-value>
    </context-param>
    <context-param>
        <param-name>com.sun.faces.verifyObjects</param-name>
        <param-value>true</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>

    <listener>
        <listener-class>
            com.sun.faces.config.ConfigureListener
        </listener-class>
    </listener>

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

    <!-- Faces Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
</web-app>
```

Context-parametern `javax.faces.DEVELOPMENT` bör enbart vara satt till true på en utvecklingsmaskin och inte när en applikation går live. URL-mappningen ser likadan ut för Facelets som för JSF – man anropar alltså inte `save_data.xhtml` – utan `save_data.jsf`.

Här kommer sista filen, `faces-config.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>
<application>
    <view-handler>com.sun.facelets.FaceletViewHandler</view-handler>
</application>

<managed-bean>
    <managed-bean-name>FaceletBean</managed-bean-name>
    <managed-bean-class>guide.facelets.FaceletBean</managed-bean-
class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>

<navigation-rule>
    <from-view-id>/save_data.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>submit</from-outcome>
        <to-view-id>/present_data.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
</faces-config>
```

Här definierar vi explicit att Facelets ska vara ViewHandler. Navigationsreglerna ser likadana ut som i föregående projekt.

Nu återstår det dock en viktig punkt – vi måste skeppa med de specifika Facelets-bibliotek som vi laddade ner tidigare, samt de fyra paket från Apache Commons. Följande filer måste placeras i katalogen `faceletsprojekt/WEB-INF/lib/`:

```
commons-beanutils.jar
commons-collections.jar
commons-digester.jar
commons-logging.jar
el-api-1.0.jar
el-impl-1.0.jar
jsf-facelets.jar
```

Nu återstår det enbart att paketera vår applikation och utplacera den (deploy) – vilket går till på samma sätt som när vi gjorde det för `jsfprojekt`. Ställ dig i rooten för projektet (delvis `faceletsprojekt`) och kör kommandot `jar -cvf faceletsprojekt.war *`
Du utplacerar sedan med hjälp av administrationssidan för Glassfish, på URLen `http://localhost:4848/`. På http://localhost:8080/faceletsprojekt/save_data.jsf hittas sedan ditt projekt.

Tredje projektet: Ett enkelt exempel på Ajax4JSF

I det avslutande projektet kommer vi se ett väldigt enkelt exempel på hur man kan arbeta med Ajax4Jsf. Vi kommer att eka tillbaka en text som användaren skriver med hjälp av Ajax. Vi kommer att använda JSF och inte Facelets i det här exemplet.

Vi börjar med vår väldigt simpla böna – en enkel POJO:

```
package guide.a4j;

public class AjaxBean {
    private String text;

    public String getText() {
        return this.text;
    }
    public void setText(String text) {
        this.text = text;
    }
}
```

Spara den som `AjaxBean.java` i `a4jprojekt/src/`, kompilera och placera `AjaxBean.class` i mappen `a4jprojekt/WEB-INF/classes/guide/a4j/`.

Vi går vidare med vårat enda klient-dokument, echo.jsp:

```
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<html>
<head>
<title>A4J Echo Text</title>
</head>
<body>

<f:view>
    <h:form>
        <h:inputText size="50" value="#{AjaxBean.text}" >
            <a4j:support event="onkeyup" reRender="repeatMe"/>
        </h:inputText>
        <h:outputText value="#{AjaxBean.text}" id="repeatMe"/>
    </h:form>
</f:view>

</body>
</html>
```

Det som händer här, är att vi använder oss av Ajax4Jsf's specifika tagg `a4j:support` till ett inputfält deklarerat med JSF – sätter ett event-attribut och anger vilket element som ska uppdateras när eventet (händelsen) inträffar. Händelsen i vårat fall är `onkeyup` – vilket innebär att så fort en tangent har tryckts in i vårat inputfält, uppdaterar vi elementet "`repeatMe`" med text från vår böna. I det här fallet ekas bara det som användaren har skrivit ut till elementet. Vi måste såklart deklarera namnrymden `a4j` för JSF, vilket vi gör högst upp i dokumentet. Spara filen som `echo.jsp` i rotkatalogen (`a4jprojekt`).

Nu återstår `web.xml` och `faces-config.xml`. Först ut är `web.xml`:

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<display-name>a4jEchoText</display-name>
<context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>server</param-value>
</context-param>
<filter>
    <display-name>Ajax4jsf Filter</display-name>
    <filter-name>ajax4jsf</filter-name>
    <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>
<filter-mapping>
    <filter-name>ajax4jsf</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
<listener>
    <listener-class>
        com.sun.faces.config.ConfigureListener
    </listener-class>
</listener>

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

<!-- Faces Servlet Mapping -->
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
</web-app>
```

Först av allt måste vi sätta ett filter för att Ajax4Jsf ska rendera sidan innan JSF gör det – annars kommer JSF inte att känna igen alla taggar och attribut, som är specifika för Ajax4Jsf.

Här kommer `faces-config.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>
    <managed-bean>
        <managed-bean-name>AjaxBean</managed-bean-name>
        <managed-bean-class>guide.a4j.AjaxBean</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
        <managed-property>
        <property-name>text</property-name>
        <value/>
    </managed-property>
</managed-bean>
</faces-config>
```

Inga nyheter i den filen.

För att få vårat projekt att snurra måste vi skeppa med de bibliotek som Ajax4Jsf kräver, och placera dom i mappen `a4jprojekt/WEB-INF/lib/`. Följande filer måste placeras där:

```
commons-beanutils.jar
commons-collections.jar
commons-digester.jar
commons-logging.jar
richfaces-api-3.2.0.GA.jar
richfaces-impl-3.2.0.GA.jar
richfaces-ui-3.2.0.GA.jar
```

Nu återstår det bara att paketera ihop projektet och utplacera det. Vi gör som i tidigare projekt och placerar oss i rotkatalogen för projektet (`a4jprojekt`) och kör kommandot:

```
    jar -cvf a4jprojekt.war *
```

Logga in i administrationsdelen för Glassfish via http://localhost:4848/ och placera ut det (deploy). Peka webläsaren till http://localhost:8080/a4jprojekt/echo.jsf.

Länksamling

Officiell sida för Glassfish: http://glassfish.dev.java.net/

Officiell sida för Facelets: http://facelets.dev.java.net/

Officiell sida för Ajax4JSF: http://www.jboss.org/jbossrichfaces/

Facelet-handbok: https://facelets.dev.java.net/nonav/docs/dev/docbook.html

Integration med Hibernate: http://www.tavutaito.fi/tutorials/jsf-facelets/