F A C H H O C H S C H U L E W E D E L

S E M I N A R A R B E I T

im Fachbereich

Wirtschaftsinformatik

Seminar:

Betriebssysteme, Werkzeuge für das Web

und Programmiersprachen

Wintersemester 2006

Thema:

JavaServer Faces

Vorstellung des Standard-Framework von Sun

Danny Falss (wi5270)

Betreuer: Prof. Dr. U. Schmidt

Inhaltsverzeichnis

1 Einführung

1.1 Der Begriff JavaServer Faces

Um sich der Materie zu nähern, soll als erstes der Begriff „JavaServer Faces“ betrachtet werden:

Java – „eine von der Firma Sun Microsystems entwickelte Softwaretechnologie“ http://wikipedia.org/

Server – „Ein Server wartet auf die Kontaktaufnahme eines Programms und tauscht nach Kontaktaufnahme mit diesem Nachrichten aus.“ http://wikipedia.org/

Faces – „die Angesichter, die Flächen, die Gesichter, die Mienen, die Zifferblätter“ http://dict.leo.org/

Also lässt sich vermuten, dass es sich bei bei JSF um eine serverbasierte Anwendung handelt, die durch/für die Java-Technologie implementiert ist. Dabei scheint sie sich im Bereich der Darstellung zu bewegen.

„JavaServer Faces (kurz: JSF) ist ein Framework-Standard im Bereich der Webanwendungen. Mit Hilfe von JSF kann der Entwickler auf einfache Art und Weise Komponenten für Benutzerschnittstellen in Webseiten einbinden und die Navigation definieren.“ http://wikipedia.org/

Genauer genommen handelt es sich bei JSF um ein Oberflächenframework, dass den/die Entwickler einer (Web-)Anwendung bei der Interaktion mit dem Benutzer unterstützt. Zum einen stellt es ein Dialoggedächnis zur Verfügung, dass über Seitenaufrufe hinaus die Zustände der Seiten speichert. Realisiert wird dieses, indem jede Interaktion des Benutzers (Klicken eines Links oder Buttons) das Abschicken eines Formulars auslöst. Zum anderen bietet es durch Strukturierungsmöglichkeiten eine gute Wartbarkeit speziell umfangreicher Projekte. Verwaltet wird das Framework dabei typischerweise durch eine zentrale XML-Datei, die faces-config.xml, die alle Einstellmöglichkeiten kapselt.

1.2 Entwicklung & Implementationen

JSF wird zum einen von Sun als Standard entwickelt, als auch nach diesem Standard implementiert angeboten, die Referenzimplementation (RI).

http://weblogs.java.net/blog/edburns/

http://itpro.nikkeibp.co.jp/free/NOS/NEWS/20021126/1/craig.jpg

Chefentwickler: Ed Burns (links) und Craig McClanahan (rechts), letzterer war auch an der Entwicklung von Struts maßgeblich beteiligt.

(**J**ava **S**pecification **R**equest = JSR)

JSR 127: JSF 1.0/1.1 Final Release 2 seit 27 Mai 2005

JSF 1.1 implementiert in

- OpenSource Projekt Apache MyFaces
- Sun RI 1.1

JSR 252: JavaServer Faces 1.2: Final Release seit 11. Mai 2006

- JSF 1.2 nur umgesetzt in Sun RI 1.2. – Teil des Applikationsservers Glassfish

Im folgenden Betrachtung der allgemeinen Technik von JSF

2 Aufbau und Techniken von JSF

2.1 Rollenkonzept

Bei der Entwicklung von JSF war ein wichtiges Ziel, dass die einzelnen Aufgaben innerhalb eines Projektes voneinander getrennt sind. Für diese Trennung wurden verschiedene Rollen eingeführt.

- Seitenautoren / Webdesigner
- Applikationsentwickler
- Komponentenentwickler
- Tool-Hersteller

Komponentenentwickler schreiben eigene JSF-Komponenten, deren Tags von Seitenautoren verwendet werden. Zusammen mit JSF-eigenen Tags und HTML entsteht dann eine komplette Seite.

Die Aufgabe der Applikationsentwickler ist es die Anwendungslogik hinter den Seiten zu implementieren. Also Daten der Seiten aufzubereiten und Events der (eigenen) JSF-Tags zu verarbeiten.

Tool-Hersteller sollen Softwarelösungen für (einfachere) Erstellen eines JSF basierten Projektes entwickeln.

2.2 Schichtenkonzept

Die Zielsetzung bei der der Entwicklung von JSF war es ein multifunktionales, aber auch leicht zu wartendes Framework zu entwickeln. Daher wurde auf Mehrschichtigkeit gesetzt,um eine klare Trennung zwischen den Aufgaben des Systems zu erreichen:

- die Präsentationschicht (Frontend) für Darstellungsaufgaben
- die Geschäftslogik und Ablaufsteuerung (Middleware)
- die Modellebene (Backend), die die Datenhaltung übernimmt.

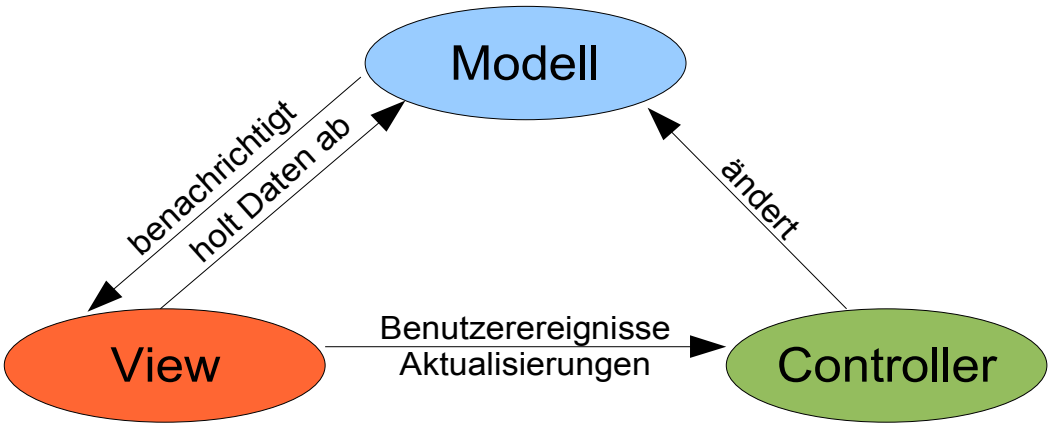
Für die Darstellung stehen zwei Tag-Bibliotheken zur Verfügung. Zum einen die Core-Bibliothek, die Standardfunktionen umfasst und zum anderen die HTML-Bibliothek, die

Tags für alle HTML-Komponenten enthält. Eine Auflistung aller Tags ist unter http://horstmann.com/corejsf/jsf-tags.html einzusehen.

JSF setzt bei der Schichtentrennung ebenso wie viele andere Frameworks auf das MVC-Controller-Konzept

2.3 Modell-View-Controller-Konzept

Das MVC-Konzept unterstützt die Aufteilung der Schichten aus dem vorherigen Kapitel.



Es wird eine Entkopplung der Elemente erreicht. Dabei ist das Modell für die Datenhaltung zuständig. Die Präsentation der Daten erfolgt durch die View und der Controller vermittelt zwischen diesen beiden Komponenten.

Das MVC-Konzept arbeitet dabei ereignisorientiert. Eine View registriert sich im Modell und wird dann bei Änderungen informiert, so dass sich die View die neuen Daten abholen kann. Sollen durch eine View Daten geändert werden, so wir ein entsprechendes Ereignis im Controller ausgelöst und der Controller aktualisiert die Daten im Modell, das dann wiederum die View benachrichtigt.

Durch die strikte Trennung soll erreicht werden, dass sich die jeweiligen Elemente nur noch mit der ihnen zugedachten Aufgabe befassen und so nicht nur die Wartbarkeit der Anwendung erhöht wird, sondern auch Wiederverwendbarkeit erreicht wird.

Ein Designer kann die Views pflegen, ohne wie bei PHP oder JSP-Seiten Codeschnipsel in die Seiten einbauen zu müssen, da er mittels der vom Framework zur Verfügung

gestellten Core- und HTML-Tags die Seite erstellt. Zusätzlich kann er noch die eigenen Erweiterungen, also eigenen Komponenten, verwenden. Lediglich die Schnittstelle zum Controller muss mit dem Applikationsentwickler abgestimmt werden. Daraus ergibt sich, dass die View-Komponente leicht austauschbar wird. Dadurch können unter identischen, logischen Abläufen und auf gleichen Daten andere Sichten erzeugt werden, z.B. eine XML Ausgabe. Unter Verwendung des gleichen Modells ist es auch denkbar eine Swing-Anwendung anstelle einer JSP-Anwendung zu erstellen, dazu ist aber auch ein neuer Controller notwendig.

Die Aufgabe des Controllers im Rahmen des MVC-Konzeptes besteht darin die Ablaufsteuerung zu übernehmen. Des weiteren kümmert er sich um Datenaktualisierungen im Modell nach Eintreffen eines entsprechenden Ereignisses. Er wird wie schon erwähnt von Applikationsentwicklern gepflegt.

Bei der Wiederverwendbarkeit ist vor allem die Verwendung von eigens erstellten Komponenten zu sehen, denn sind diese einmal erstellt, so können sie in jeder Anwendung verwendet werden. Auch Modelle sind wiederverwendbar, aber Views und Controller meistens nur gemeinsam aus der vorher schon erwähnten engen Verbindung.

Nachteile der strikten Trennung sind ein erhöhter Aufwand bei der Erstellung und der Ablauf ist nicht immer sofort zu erkennen, wodurch das Verständnis erschwert wird.

Ausgehend von der Schichtentrennung lassen sich das Frontend bzw. Backend im MVC-Konzept eindeutig der View bzw. dem Modell zuordnen und die Ablaufsteuerung sowie Aktualisierung der Daten übernimmt der Controller. Die Geschäftslogik der Anwendung kann entweder im Modell oder im Controller untergebracht werden. Bei umfangreichen Projekten wird diese sogar oft komplett ausgelagert, so dass sie sauber getrennt von JSF abläuft. Mittels EJBs (Enterprise Java Beans) können Transaktions-, Namens- oder Sicherheitsdienste realisiert werden, sie bilden einen eigenständigen Teil. Generell empfiehlt es sich für die Geschäftslogik das Entwurfsmuster der Fassade zu verwenden, da dadurch die Schnittstelle zu der Geschäftslogik leicht durch neue Funktionen erweitert werden kann bzw. vorhandene Funktionen transparent ausgetauscht werden können.

2.4 JSF Expression Language

Die Verknüpfung zwischen Views und dem Controller erfolgt mittels der JSF EL. Damit ist es möglich innerhalb einer View auf Methoden zu zugreifen, dieses wird z.B. für Listener benötigt und nennt sich Method-Binding. Auch der Zugriff auf Variablen von Klassen kann erfolgen, so genanntes Value-Binding. JSF EL Ausdrücke sehen dabei so aus: `#{Klasse.ziel}`. Die Übergabe von Parametern ist dabei nicht möglich.

2.5 Bean-Management, Managed-Beans und Backing-Beans

JavaBeans sind Java Klassen, die zu Speicherung von Benutzer- oder Anwendungswerten verwendet werden. Die Verwaltung der Klassen beziehungsweise genauer die Verwaltung der Objekte übernimmt JSF. In JSP wurden diese JavaBeans in jeder Seite eingebunden. Managed-Bean bezeichnet dabei das Verfahren dieser (globalen) Bereitstellung, sprich die Verwaltung der Beans. Die Inkarnation erfolgt nach dem Lazy loading Prinzip. D.h. sie erfolgt immer erst dann, wenn auf eine Bean zum ersten Mal zugegriffen wird. Dazu kann zentral über die Konfigurationsdatei gesteuert werden, welchen Gültigkeitsbereich die erzeugten Objekte haben:

- Keine Instantiierung, nur Bekanntmachung der Klasse
- application: als Singleton, Anwendungsglobal
- session: als Objekt innerhalb jeder Sitzung, also über Seitenwechsel hinaus
- request: als Objekt nur gültig für diesen Request (Seite)

```
<managed-bean>
   <description>Verwaltet alle Teilnehmer</description>
   <managed-bean-name>
         MemberControllerBean
   </managed-bean-name>
   <managed-bean-class>
         sf.controller.MemberController
   </managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Vgl. dazu Beispiel /Webroot/WEB-INF/tutadmin/faces-config.xml

Über ValueBinding können diese JavaBeans mit Komponenten verknüpft werden, so dass eine Kopplung zwischen dem Modell und der Sicht entsteht. Die verknüpften Beans heißen dann Backing-Beans. Sie gehören aus JSF-Sicht zur Schicht des Modells und werden auch als Modellobjekte bezeichnet. Sie besitzen Getter- und Setter-Methoden für ihre Membervariablen und können auch Aktions- sowie Validierungsmethoden bereitstellen. Allerdings ist es sauberer die Methoden in einen Controller zu kapseln, um das Modell möglichst unabhängig zu halten.

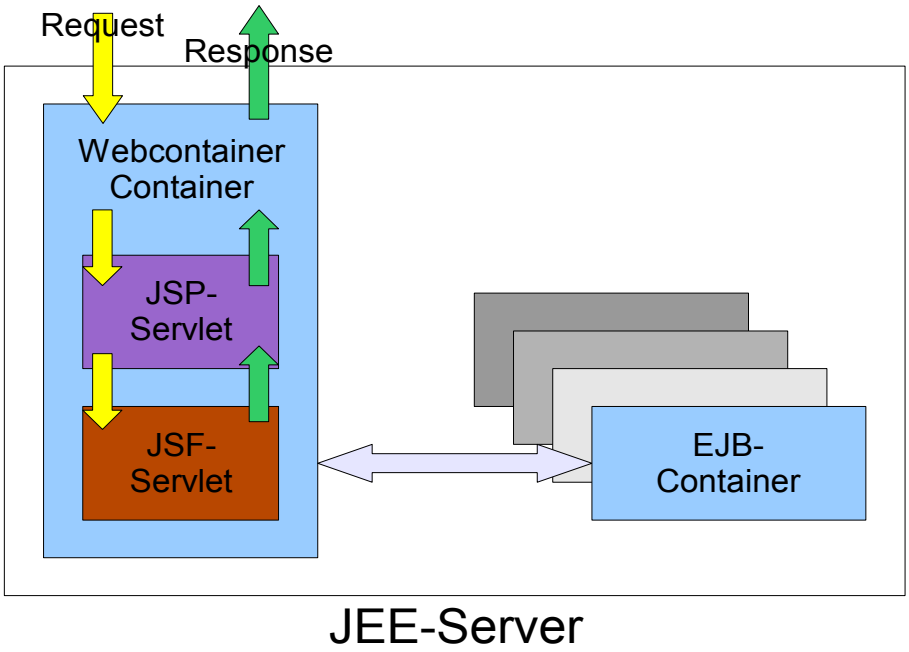
```
<h:inputText id="name"
   value="#{MemberControllerBean.currentMember.name}">
```

Vgl. dazu Beispiel /Webroot/tutadmin/member_edit.jsp

In MemberControllerBean wird auf eine Variable namens `currentMember` zugegriffen von der wiederum die Eigenschaft `name` interessiert. Der interne Zugriff erfolgt über die Getter- und Setter-Methoden: `.getCurrentMember.getName()` bzw. `.getCurrentMember.setName(neuerWert)`.

2.6 Überblick über JEE-Server

Das JSF-Framework läuft innerhalb des JEE-Servers als Servlet, also ein serverseitiges Applet. Anfragen die den Server erreichen werden über das JSP-Servlet an das JSF-Servlet weitergeleitet und dieses kann die Antwort erstellen.



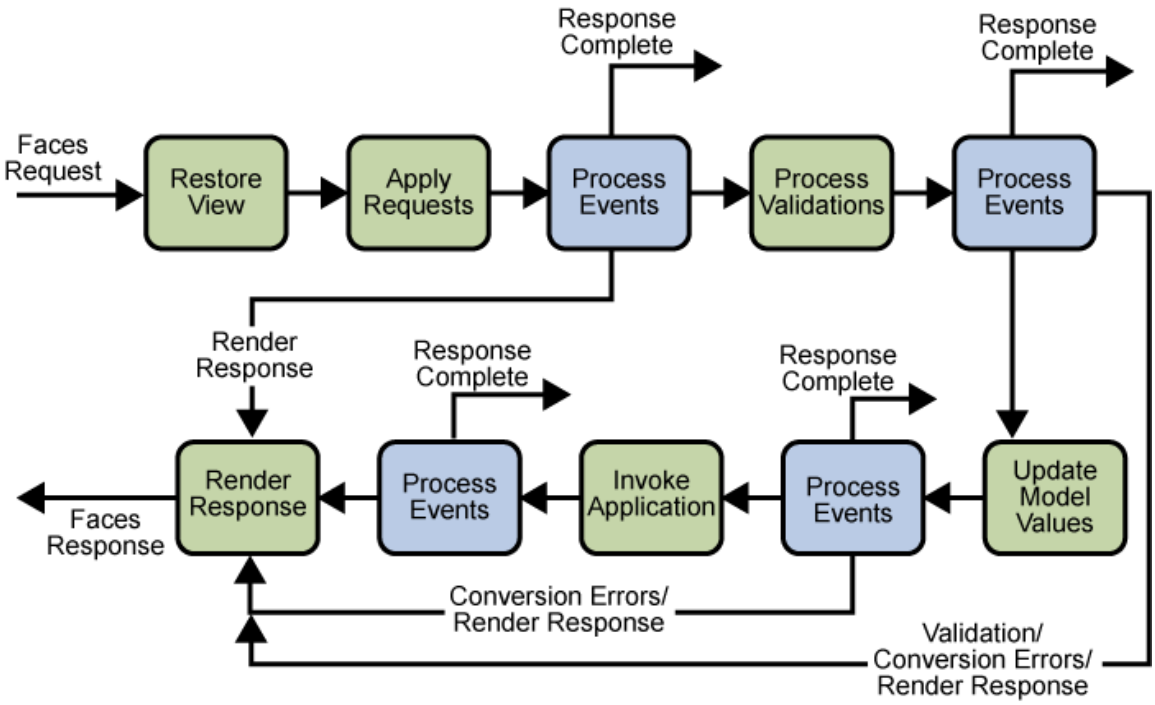
Bekannte Applikation Server für Java-Anwendungen sind zum einen Glassfish von SUN (auf Basis des Java-Application-Servers) sowie Apache Tomcat. Um aus einer entsprechenden Anfrage (Request) eine Antwort zu generieren (Response) existiert innerhalb des Frameworks ein gewisser Grundmotor, der durchlaufen wird: der Lebenszyklus

2.7 JSF Lifecycle

In Abhängigkeit von der Request- bzw. Responseart werden verschiedene Aktionen von Framework ausgeführt. Dabei werden bei der Arbeit mit JSF generell je zwei Arten von Request (Anfrage) und Response (Antwort) unterschieden:

- Faces-Request: die Abfrage einer JSF-Seite
- Non-Faces-Request: alle anderen Anfragen (JSP, statisches HTML etc.)
- Faces-Response: Antwort mit einer JSF-Seite
- Non-Faces-Response: alle anderen Antworten (JSP, statisches HTML etc.)

Die Requests und Responses können in beliebiger Kombination auftreten, so kann es vorkommen, dass von einer reinen HTML-Seite (Non-Faces-Request) zu einer Faces-Seite (Faces-Response) verwiesen wird. Dieses ist meistens dann der Fall, wenn die entsprechende Anwendung gestartet wird. Umgekehrt folgt beim Verlassen einer Faces-Anwendung einem Faces-Request eine Non-Faces-Response. Der Standardfall bei der Arbeit mit JSF ist der Faces-Request gefolgt von einer Faces-Response. Dieses wird auch der Standard Lebenszyklus eines Request genannt und soll hier nun genauer betrachtet werden, da an diesem Ablauf viele Eigenschaften von JSF erklärt werden können.



http://java.sun.com/javaee/5/docs/tutorial/doc/images/jsfIntro-lifecycle.gif

Restore View

Um das Eingangs erwähnte Dialoggedächtnis zu realisieren ist es notwendig die (Formular)Zustände zu speichern und wiederherzustellen. Dabei werden zwei Fälle unterschieden:
Beim erstmaligen Abruf einer Seite existiert noch kein View-Objekt. Daher muss eines erzeugt werden, in dem ein leerer Komponentenbaum angelegt und im FacesContext einhängt wird. Danach kann sofort zur Render Response-Phase gesprungen, da keine Verarbeitung von übergebene Werten etc. nötig ist.
Sollte bereits ein Komponentenbaum im FacesContext existieren, so wird dieses View-Objekt geladen und mit Validatoren, Konvertern und Listenern verknüpft.

Apply Request Values

In dieser Phase werden die übertragenen Werten des Requests (also die Daten des abgeschickten Formulars) in den Komponentenbaum übernommen, also in den entsprechenden Komponenten gesetzt. ActionEvents werden hier generiert, z.B, das Drücken des Buttons durch den der Request erzeugt wurde. Dabei ist wichtig dass hier noch keine Änderung des Modells erfolgt, sondern nur die String-Werte vorbereitet werden.

Process Validation

Anhand von Konvertern (aber auch Renderern) werden die gespeicherten Werte der vorherigen Phase in die Zielformate (Modelldatentypen) überführt. Anschließend werden alle Werte der Komponenten mittels registrierter Validatoren überprüft. Im Fehlerfall werden dabei üblicherweise komponentenbezogene Meldungen generiert und es wird mit der Render Response-Phase fortgefahren. Wodurch dieselbe Seite gerendert wird, da ihr Komponentenbaum noch FacesContext liegt.

Update Model Values

Wenn bis hier hin kein Fehler auftrat, so werden die überprüften Werte in das Modell übernommen. Dabei werden ValueChangeEvents generiert, falls sich ein Wert geändert hat. Nach dieser Phase werden die registrierten Listener über Wertänderungen informiert.

Invoke Application

Alle Ereignisse der Anwendungsebene werden verarbeitet. So wird z.B. die nachfolgende Seite ermittelt und ihr Komponentenbaum im Kontext abgelegt. Nach dieser Phase werden alle registrierten ActionListener benachrichtigt.

Render Response

Der im Kontext befindliche Komponentenbaum wird ausgegeben. Dazu wird die encode-Methode jeder Komponente ausgeführt. Wird also die Phase Invoke Application übersprungen so erfolgt das Rendern derselben Seite.

Process Events

Zum einen werden die für Events registrierte Listener benachrichtigt, als auch PhaseListener benachrichtigt. Letztere können den Ablauf beeinflusse oder nebenläufige Tätigkeiten angestoßen (z.B. Logging). Generell wird bei schweren Fehlern die Verarbeitung abgebrochen (Response Complete) oder vorzeitig eine Ausgabe erzeugt (Render Response).

2.8 Das Dialoggedächtnis mittels Zustandsspeicherung (State Saving)

Es kann in der Konfigurationsdatei ausgewählt werden ob Clientseitig mittels Serialisierung und Hidden-Fields in Formularen oder Serverseitig mittels einer Session die Zustände gespeichert werden sollen. Nachteil von serverseitiger Speicherung ist das Problem des Timeouts nach einer bestimmten inaktiven Zeit, sowie die potentielle Angreifbarkeit des Systems z.B. durch das Senden einer großen Anzahl von Anfragen an den Server (für jede Anfrage wird eine Session eröffnet und Resourcen verbraucht). Das Speichern beim Client hingegen erhöht den Datenverkehr erheblich, da die gesamten Sitzungsdaten immer in den Seiten codiert vorliegt. Außerdem ergibt sich zudem die Gefahr, dass ein bösartiger Benutzer es schafft die Session zu verändern, z.B. ein Login vorzutäuschen.

2.9 Validierung und Konvertierung

Wie beschrieben müssen Daten vor der Übernahme in das Modell umgewandelt und überprüft werden. Dabei ermöglichen es Validatoren Stringeingaben einer Bereichsprüfung zu unterziehen. Konverter werden von JSF bei ValueBindings implizit ausgeführt, können aber auch explizit angegeben werden.

JSF liefert drei Standardvalidatoren:

- validateLength: Angabe einer Mindest- und Maximallänge.
- validateLongRange: Bereichsprüfung von ganzen Zahlen
- validateDoubleRange; Bereichsprüfung von Fließkommazahlen

Und darüber hinaus noch etliche Konverter für das Umwandeln von Strings aus Eingabekomponenten in die entsprechenden Java-Datentypen u.U.:

Id	Klasse
javax.faces.BigDecimal	javax.faces.convert.BigDecimalConverter
javax.faces.BigInteger	javax.faces.convert.BigIntegerConverter
javax.faces.Boolean	javax.faces.convert.BooleanConverter
javax.faces.Byte	javax.faces.convert.ByteConverter
javax.faces.Character	javax.faces.convert.CharacterConverter
javax.faces.DateTime	javax.faces.convert.DateTimeConverter
javax.faces.Double	javax.faces.convert.DoubleConverter
javax.faces.Float	javax.faces.convert.FloatConverter

2.10 Navigationskonzept

Nach der Übernahme der Daten erfolgt das Ausführen der eigentlichen Anwendung. In diesem Rahmen ist auch die Ablauflogik zu sehen. Die Verknüpfung der einzelnen Views ist nicht statisch in der jeweiligen Seite unterzubringen, sondern die Verkettung erfolgt über die Konfiguartionsdatei. Es werden also Symbole eingeführt die innerhalb der Views verwendet werden, so dass ein umdefinieren eines Symbols Auswirkungen auf alle Seiten hat. Dieses ist interessant für Portalseiten. Durch Navigationsregeln wird festgelegt zu welcher Seite verwiesen werden soll, wenn eine bestimmte Aktion einen bestimmten Rückgabewert (Outcome) liefert.

```
<navigation-rule>
   <from-view-id>/tutadmin/member.jsp</from-view-id>
   <navigation-case>
         <from-outcome>ta_member_edit</from-outcome>
         <to-view-id>/tutadmin/member_edit.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

Vgl. dazu Beispiel /Webroot/WEB-INF/tutadmin/faces-config.xml

Von der Sicht member.jsp ist die nächste anzuzeigende Seite member_edit.jsp, falls der Rückgabewert von member.jsp ta_member_edit ist. Eine spezielle Ausgangsaktion wurde hierbei nicht angegeben. Die Angabe könnte mir `<from-action></from-action>` erfolgen. Ein Link würde dann so aussehen: `<h:commandlink action="ta_member_edit" value="Editieren"/>`

Dynamik kommt ins Spiel, wenn nicht statisch das Symbol `ta_member_edit` verwendet wird, sondern an dieser Stelle eine Methode aufgerufen wird, die einen entsprechenden Navigationsstring zurückliefert. Wie schon erwähnt erfolgt aufgrund dieses Strings (also das Symbol) die Ermittlung der Folgeseite.

2.11 Internationalisierung

In Zeiten globalen Wettbewerbs werden nahezu alle größeren Softwarelösungen international eingesetzt. Wünschenswert wäre es, wenn sich die Webseite an die jeweiligen Spracheigenschaften bzw. -wünsche des Benutzers anpassen würde. JSF nutzt hierzu die etablierten Resource Bundles aus Javas I18N- und L10N-APIS.

Drei Schritte zur Internationalisierung:

1. Anlegen einer Resourcedateien pro Sprache.

   Vgl. dazu Beispiel /src/sf/resource/msg_de.properties, msg_en.properties

2. Eintragen der Resourcedatei und der verfügbaren Sprachen in die Konfigurationsdatei

```
<application>
      <locale-config>
            <supported-locale>de</supported-locale>
            <supported-locale>en</supported-locale>
      </locale-config>
      <message-bundle>sf.resource.msg</message-bundle>
</application>
```

   Vgl. dazu Beispiel /Webroot/WEB-INF/tutadmin/faces-config.xml

3. Benutzen der Resourcedateien

```
<f:view locale="#{LocaleControllerBean.currentLocale}">
```

   Darstellen der aktuellen View mit der ausgewähtlen Sprache

```
<f:loadBundle basename="sf.resource.msg" var="res" />
```

   Bekanntmachen des Resourcebundles innerhalb dieser Datei unter der Variable `res`

```
<h:outputText value="#{res.title}"/>
```

   Ausgabe des Titels `title`

   Vgl. dazu Beispiel /Webroot/tutadmin/overview.jsp

2.12 Events & Listener

Während der Abarbeitung der verschiedenen Phasen werden von JSF bzw. von den Komponenten Ereignisse (Events) generiert und in Warteschlangen eingereiht. Am Ende der Phasen werden die entsprechenden Events abgearbeitet und die verknüpften Methoden (Listener) aufgerufen.

1. ActionEvent und -Listener

Ereignisse werden generiert in der Apply Request Values-Phase. Die Verarbeitung bzw. Übergabe an die Listener erfolgt erst in der Invoke Application-Phase. ActionListener können an CommandButtons und CommandLinks angehängt werden. Für Navigationszwecke wird das `action`-Attribut verwendet. Methodenaufrufe, z.B. das Setzen eines Zählers können mittels des `actionListener`-Attributs oder des `<f:actionListener id="”"/>`-Tags erfolgen. Mit dem Tag können auch mehrere Listener eingehängt werden.

2. ValueChangeEvent und -Listener

Wenn sich ein Wert im Modell ändert (Phase: Update Model Values) werden die Ereignisse erstellt und anschließend an diese Phase von den Listenern verarbeitet. ValueChangeListener können an alle UIInput-Komponenten angehängt werden. Dazu existiert ähnlich wie bei den ActionListenern zum einen das `valueChangeListener`-Attribut und das `<f:valueChangeListener id="”"/>`-Tags

3. PhaseEvent und -Listener

Wird ein PhaseListener erstellt, so wird dieser vor und nach einer oder allen Phasen des Lebenszyklus aufgerufen und seine Methode gestartet. Das Beobachten aller Phasen kann bei Debuging, Monitoring oder Logging interessant sein. Eine einzelne Phase zu überwachen kann z.b. für das Überprüfen eines Logins sinnvoll sein (dazu wird dann vor der RestoreView-Phase eine Prüfung durchgeführt)

Bei der Verwendung von PhaseListenern sowie von `actionListener` bzw. `valueChangeListener`-Tags müssen diese entsprechend in der Konfiguaration eingetragen werden.

3 Eigene Erweiterungen

3.1 Eigene Komponenten

3.1.1 Schritte bei der Erstellung eigener Komponenten

- Komponenten Tag-Handler entwickeln:
  Verwaltet Attribute und liefert den Renderer der Komponente und den Typ
- TagLibaryDescriptor (TLD) erstellen:
  XML Datei die die Verwendung des Tags beschreibt.
- Komponentenklasse entwickeln:
  die Funktionalität der Komponente
- zugehörige Renderklasse entwickeln
  bestimmt das Aussehen der Komponente
- Bekanntmachung in Konfigurationsdatei
- Benutzen der TLD bzw. Komponente

3.1.2 Validator

Natürlich lassen sich eigene Validatoren bzw. Konverter über das Implementieren der Validator- bzw. Converter-Schnittstelle erstellen. Es ist auch möglich Validatoren direkt innerhalb einer Controller-Klasse zu schreiben, dieses widerspricht aber dem Trennungsansatz und daher sollte darauf verzichtet werden.

Im folgenden soll anhand zur Überprüfung von Emailadressen ein Validator erstellt werden. Dazu wird eine Java-Klasse erzeugt, die das Interface Validator implementiert. Die vorgegebene Methode ist:

```
public void validate(FacesContext context, UIComponent
component, Object value) throws ValidatorException
```

Man erhält also sowohl den aktuellen Context `FacesContext` als auch die zu überprüfende Komponente `UIComponent` und den aktuellen Wert `value` der Komponente. Im Fehlerfall wird eine `ValidatorException` ausgelöst mit Angabe der Fehlernachricht `FacesMessage`.

Vgl. dazu Beispiel /src/util/validator/Mail.java

Der so erstellte Validator muss dann in die Konfiguration eingetragen werden:

```
<validator>
      <validator-id>MailValidator</validator-id>
      <validator-class>
            util.validator.Mail
      </validator-class>
</validator>
```

Vgl. dazu Beispiel /Webroot/WEB-INF/tutadmin/faces-config.xml

Und kann dann an eine Komponente angehängt werden mittels dem `<f:validator/>`-Tag:

```
<h:inputText id="email"
      value="#{MemberControllerBean.currentMember.email}"
      required="true" maxlength="50">
      <f:validator validatorId="MailValidator"/>
</h:inputText>
```

Vgl. dazu Beispiel /Webroot/tutadmin/member_edit.jsp

3.1.3 Fortschrittsanzeige

(Quelle: Bosch, Andy – um JSF EL Funktionalität erweitert)

Es soll eine Komponente entwickelt werden, die sowohl grafisch als auch in Worten eine Prozentangabe ausgibt. Die Komponente soll `UsageBar` heißen und der hinterlegte Wert soll in `inuse` gespeichert werden. Dabei wurde die Komponente erweitert, so dass sie auch mit Ausdrücken der JSF EL benutzt werden kann.

- Komponenten Tag-Handler entwickeln

  Der Komponenten Tag-Handler wird angesprochen, wenn ein Tag verwendet wurde. Während der Apply Request Value-Phase wird in ihm der Stringwert der Attribute gespeichert. Weiter hin liefert der Tag-Handler Informationen über die Komponentenklasse `getComponentType()` und über den zuständigen Renderer mittels `getRendererType()`. Für die Klasse `UsageBarTag` ist das Attribut `inuse` einzurichten, das den anzuzeigenden Prozentwert enthält. Dabei muss aber beachtet werden, dass sowohl ein direkter (fester) Wert, aber auch ein Laufzeitwert, angegeben werden kann.

  Vgl. dazu Beispiel /src/com/edu/jsf/bsp/tag/UsageBarTag.java

- TLD schreiben

  Der im Schritt zuvor entwickelte Tag-Handler wird mittels XML beschreiben, damit er auch in einer Seite verwendet werden kann.

  Vgl. dazu Beispiel /WebRoot/WEB-INF/custom-jsf-tld

- Komponenten-Klasse entwickeln

  Alle eigenen Komponenten erben von `UIComponentBase`, wobei spezieller auch von `UIInput`, `UIOutput` oder `UICommand` geerbt werden kann, dementsprechend stehen dann weitere Funktionen zur Verfügung (z.B. Ereignisverarbeitung). Dieses erfordert dann das implementieren weiterer Schnittstellen. Generell übernimmt die Komponentenklasse die Validierung und Zustandsspeicherung sowie das Modellupdate und die Ereignisverarbeitung. Ebenso kann auch das Rendern in dieser Klasse erfolgen. Aus Demonstrationszwecken wurde hier aber eine separate Render-Klasse angelegt. Die Komponenten-Klasse `UIUsageBar` stellt daher nur die Getter und Setter für `inuse` bereit, wobei beim Get darauf zu achten ist, einen vorliegenden Value-Bind aufzulösen. Zusätzlich wird mittels `saveState` und `restoreState` das Speichern und Wiederherstellen des vorherigen Zustandes unterstützt. Bei serverseitiger Zustandsspeicherung werden diese Methoden nicht benötigt, aber bei clientseitiger Speicherung würde der zugeordnete Wert nicht mit gespeichert werden und entsprechend verloren gehen.

  Vgl. dazu Beispiel /src/com/edu/jsf/bsp/tag/UIUsageBar.java

- Die Renderklasse `UsageBarRenderer`

  Renderer beerben die Klasse `Renderer` und kümmern sich um die (De-) Codierung. Das Decodieren bedeutet in diesem Zusammenhang, dass Werte aus einem Request in die Komponente übernommen werden können. Beim Codieren werden drei Methoden zeitlich nacheinander abgearbeitet: `encodeBegin`, `encodeChildren` und `encodeEnd`. Somit kann der Start, der Body, also die Kinder und schließlich das Ende der Komponente gerendert werden. Für die Prozentanzeige genügt es `encodeBegin` zu verwenden, da kein Body akzeptiert wird.

Vgl. dazu Beispiel /src/com/edu/jsf/bsp/tag/UsageBarRenderer.java

- Schlussendlich müssen Render- und Komponentenklasse nur noch in die Konfiguration eingetragen werden und können dann wie die JSF-Standard-Tags durch Angabe der TLD verwendet werden.

  Vgl. dazu Ende von Beispiel /Webroot/WEB-INF/tutadmin/faces-config.xml

3.2 PhaseListener

Bei Implementierung des `PhaseListener`-Interface müssen folgende drei Methoden angelegt werden:

```
public PhaseId getPhaseId()
```
Rückgabewert legt fest, ob eine bestimmte oder alle Phasen überwacht werden sollen. (Auswahl erfolgt durch Konstanten z.B `PhaseId.ANY_PHASE`)

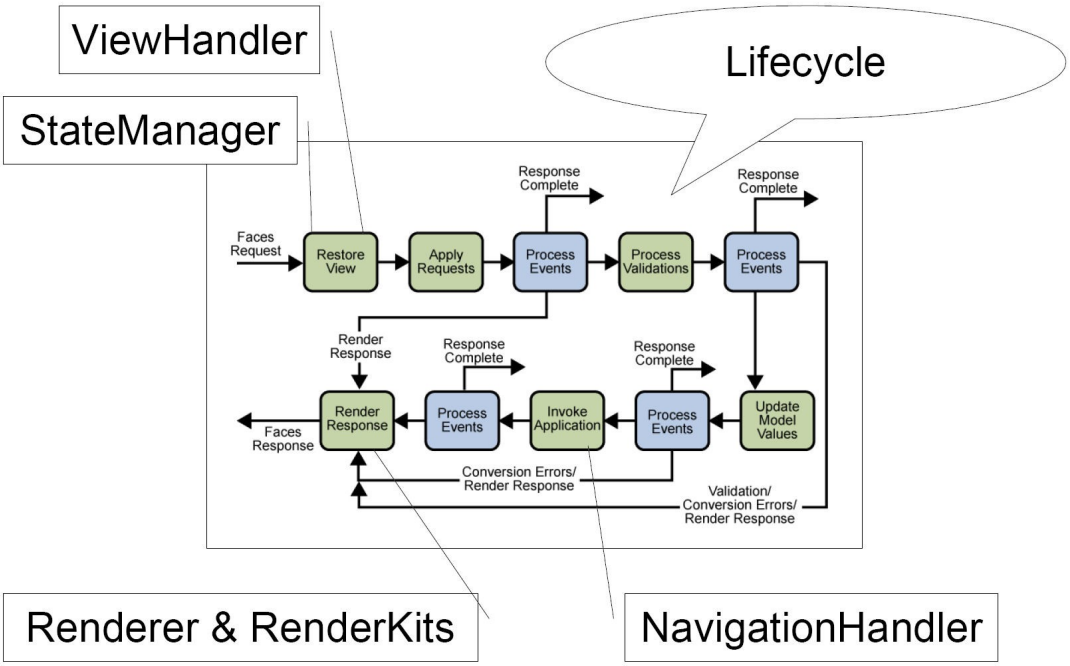
```
public void beforePhase(PhaseEvent pE)
```
wird vor der Phase ausgeführt

```
public void afterPhase(PhaseEvent pE)
```
wird nach der Phase ausgeführt

Vgl. dazu Beispiel src/util/listener/AllPhaseListener.java

Mittels PhaseListenern ist es möglich Logging zu realisieren in dem die erwähnte `PhaseId.ANY_PHASE` verwendet wird (dieses macht der Beispiel PhaseListener). Ein anderes Einsatzgebiet wäre das Kontrollieren von kritischen Parametern z.B. für Adminbefehle (vor `PhaseId.RESTORE_VIEW`). Also eine Kontrolle, ob eine Zugriffsberechtigung wirklich vorliegt.

3.3 Angepasster Lifecycle



Der Standard Lebenszyklus kann komplett angepasst werden. So kann der komplette Ablauf ausgetauscht werden oder nur einzelne Elemente. Innerhalb der Restore View Phase kann der `StateManager` zum Verwalten der Wiederherstellung und Speicherns sowie der `ViewHandler` zum Verwalten der Sichten angepasst werden. Dazu werden die entsprechenden Klassen beerbt. Gleiches ist möglich, wenn man das Navigationsverhalten verändern bzw. erweitern will. Hierzu kann die Klasse `NavigationHandler` erweitert werden. Wie schon erwähnt werden in der Render Response Phase alle Komponenten gerendert. Dazu können für einzelne Komponenten eigene Renderer erstellt werden oder für alle Komponenten komplette RendererKits angelegt werden.

4 Fazit

4.1 Vor- & Nachteile

Vorteile	Nachteile
Große Projekte: Strukturierung erhöht Übersichtlichkeit und ermöglicht Wiederverwendung in anderen Projekten	Kleine/Einzel Projekte: Zuviel „Overhead“
Volle Kontrolle, komplett anpassbar	Viel Anfangsarbeit LOC ...
„Zusammklick Anwendung“ *VISION*	Erfordert aktiviertes Java-Script

Die Entwicklung einer Anwendung mittels JSF ist dann ratsam, wenn es sich zum einen um ein umfangreiches Projekt handelt und zusätzlich zu erwarten ist, dass das komplette Projekt oder wenigstens Teile davon später für kommende Projekte wiederverwendet werden können. Für Anwendungsentwicklungsfirmen kann die Arbeit mit dem JSF-Framework daher interessant sein, da einmal entworfene Komponenten für alle Kunden eingesetzt werden können. Zusätzlich sind durch die Rollenverteilung gute Strukturierungsmöglichkeiten vorhanden. Für Einzelprojekte oder kleinere Anwendungen scheint JSF jedoch aufgrund des großen Anfangsaufwands eher ungeeignet.

Der optimistische Ansatz, dass Anwendungen durch entsprechende Tools und mittels umfangreicher Komponentendatenbanken nur noch „zusammengeklickt“ werden können scheint noch in weiter Ferne zu sein. Das Produzieren von länglichem Code ist bei Java bekannt, viel Tipparbeit wird allerdings von Entwicklungsumgebungen wie Eclipse abgenommen. Obwohl aktiviertes JavaScript heutzutage zum Standard bei Browsern gehört, kann es deaktiviert werden, wodurch eine JSF-Anwendung nicht mehr lauffähig sein würde.

4.2 Interaktion mit anderen Techniken/Frameworks

Das Oberflächenframework JSF kommt ohne weitere Funktionalität, kann aber durch die Verwendung anderer Frameworks leicht ergänzt werden. Im Folgenden werden für Aufgabenbereiche bekannte Frameworks vorgeschlagen.

- Datenbankanbindung

Mittels Apache Torque oder Hibernating

- Ajax

Ajax4JSF Framework liefert Ajax Komponentenerweiterung der JSF Komponenten

- Spring

Das Framework JSF-Spring liefert eine Anbindung von JSF an Spring und auch Spring unterstützt von Haus aus die Verwendung von JSF zur Darstellung.

- Struts-Anwendungen

bietet ergänzende Aktionselemente und ist sehr präsent. Leichtere Migration von Struts zu JSF-Anwendungen durch Einsetzen beider Frameworks.

- Layout

Facelets Framework für Templating und Tobago für „Desktop-look-and-feel Anwendungen“

5 Literaturverzeichnis

5.1 Online

http://de.wikipedia.org/

Begriffserklärungen, Einstieg in die Materie

http://www.pdbm.de/

Webseite von Prof. Dr. Bernd Müller (siehe auch Bücher)

http://horstmann.com/corejsf/

Tagauflistung

http://www.jamesholmes.com/JavaServerFaces/

Sammlung an Artikeln und Blogs zu JSF

http://jsftutorials.net/

weitere Artikel und Beispiele zu JSF

http://java.sun.com/javaee/javaserverfaces/

Einstiegsseite bei SUN

http://java.sun.com/javaee/5/docs/tutorial/doc/index.html

SUN JEE Tutorial

5.2 Bücher

JavaServer Faces, Ein Arbeitsbuch für die Praxis

Prof. Dr. Bernd Müller, Hanser Verlag, 2006, ISBN 3-446-40677-8

Java Server Faces, Das Standard-Framework zum Aufbau webbasierter Anwendungen

Bosch, Andy, Addison-Wesley, 2004, ISBN 3-827-32127-1