



Alles Ansichtssache

Inside JavaServer Faces

Gerd Beneken, Florian Deißböck

Die Spezifikation der JavaServer Faces (JSF) ist nun weitgehend abgeschlossen. Der Proposed Final Draft der JSF 1.0-Spezifikation wurde Ende letzten Jahres veröffentlicht und hat glücklicherweise einige der Ungereimtheiten der Vorversionen beseitigt. Dieser Artikel beschreibt Architekturaspekte der JavaServer Faces.

Bei der Lektüre der JavaServer Faces-Spezifikation ist einer der Autoren unschwer erkennbar: Craig McClanahan. Er ist der Erfinder des Web-Frameworks Struts und gleichzeitig der Co-Specification Lead der JavaServer Faces. Konzeptionell kann man JavaServer Faces als eine Mischung aus dem Web-Framework Struts und Java-Swing bezeichnen.

Dieser Artikel beschreibt den Proposed Final Draft der JSF-Spezifikation 1.0 [JSF03], er ist eine Aktualisierung und Vertiefung von [Zau04]. Die Beispiele wurden mit der Referenzimplementierung JSF 1.0 Beta von Sun getestet. Änderungen in Spezifikation und Code sind bis zur endgültigen Version weiterhin möglich.

Was sind JavaServer Faces?

Der folgende Quelltext soll einen ersten Eindruck von den JavaServer Faces vermitteln. Er zeigt JSP-Code, der eine Eingabeseite für Vornamen und Nachnamen erzeugt. Dafür werden die JSF-TagLibs verwendet:

```
<HTML>
<HEAD><TITLE>UserDemo</TITLE></HEAD>

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

<BODY BGCOLOR="white">

<f:view>
<h:form>

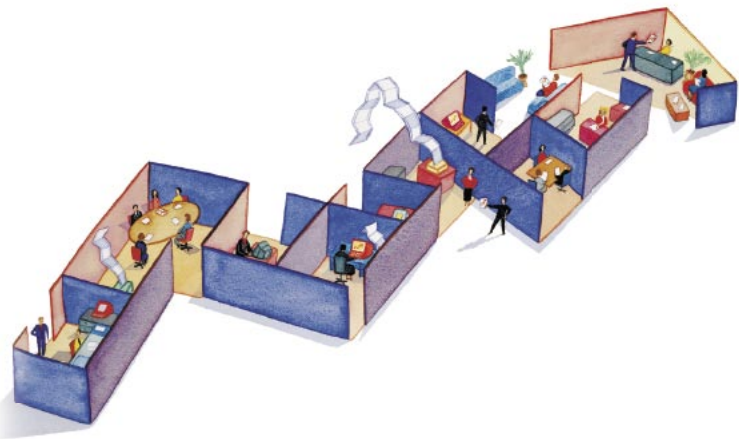
Vorname:
<h:input_text id="vorname" value="#{user.vorname}"/> <br>
Nachname:
<h:input_text id="nachname" value="#{user.nachname}"/> <br>

<h:command_button action="success" value="WELCOME">
<f:action_listener type="WelcomeActionListener"/>
</h:command_button>

</h:form>
</f:view>
</BODY>
</HTML>
```

Jedes GUI-Element wird durch ein JSF-Tag dargestellt: Die HTML-Form wird mit `<h:form>` erzeugt. Darin befinden sich zwei Eingabefelder, die mit `<h:input_text>` angegeben werden und ein Submit-Button. Dieser wird mit `<h:command_button>` erzeugt. Tags zur Erzeugung von GUI-Elementen gibt es auch bei Struts und anderen Web-Frameworks. Was ist bei Java-Server Faces anders?

Üblicherweise erzeugen JSP-Tags direkt den HTML-Code, wie er auf der Seite dargestellt wird. Am Server gibt es keine Repräsentation der Oberflächenelemente. JavaServer Faces arbeiten anders: Sie erzeugen



über die Tags *serverseitig* einen GUI-Komponentenbaum (`javax.faces.component.UIComponent`), welcher die Oberfläche repräsentiert. Java-Swing modelliert Oberflächen mit seinen `JComponents` ähnlich.

Am Server gibt es für jedes Oberflächenelement ein Objekt einer `UIComponent`-Implementierung (z. B. `UIForm`, `UICommand` oder `UIInput`). Dieses Objekt hält serverseitig den Zustand des GUI-Elements, z. B. die Daten eines Eingabefeldes. Das Objekt ist für die grafische Darstellung (z. B. in HTML) verantwortlich. Die Darstellung wird in der Regel an einen separaten Renderer delegiert.

Im Gegensatz zu den Vorversionen von JSF [EA4] stehen nun auch Container-Komponenten wie `UIPanel` zur Verfügung, die eigenständig für das Layout ihrer Subkomponenten sorgen. Dadurch kann zum Grossteil auf HTML-Layout-Mechanismen wie Tabellen verzichtet werden. Das Standardproblem von JSP-basierten Anwendungen des kaum wartbaren HTML/Java-Mixes wird damit deutlich gemildert.

Abbildung 1 zeigt die Erzeugung des Komponentenbaums, welcher von der JSP definiert wird.

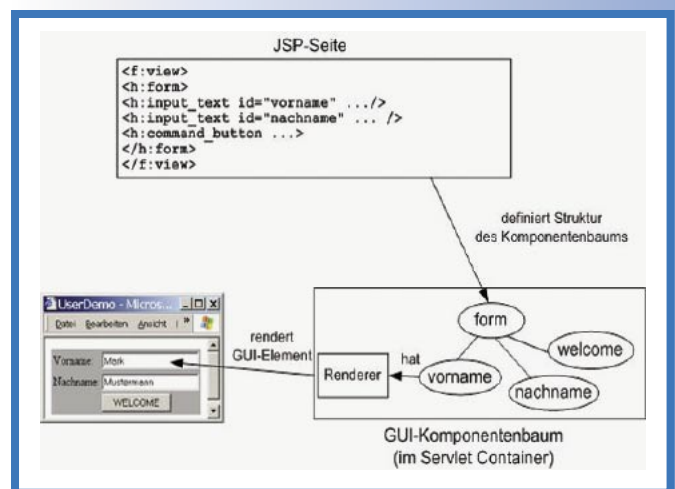


Abb. 1: JSF GUI-Komponentenbaum und seine Erzeugung

Grundideen der JavaServer Faces

Durch ihre Rahmenarchitektur sind die JavaServer Faces in der Lage, die Programmierung von Web-Oberflächen zu vereinfachen. Folgende Grundideen werden verfolgt:

- Die JSF stellen GUI-Komponenten zur Verfügung. Das sind derzeit einfache Elemente wie Label, Buttons, Eingabefelder und Check-boxen. Komplexere Elemente wie Trees und Tabellen werden in den mitgelieferten Beispielen definiert. Erweiterungen über eigene GUI-Komponenten sind möglich.



- ▼ Das Groblayout der Oberfläche kann über GUI-Komponenten festgelegt werden.
- ▼ Die GUI-Komponenten sind für jede JSP-Seite als Baumstruktur am Server vorhanden. Sie definieren das Aussehen der Elemente auf der Web-Seite, dazu werden in der Regel Renderer verwendet.
- ▼ Ein RenderKit zur Erzeugung von Standard-HTML 4.01-Code ist vorhanden. Eigene Renderer für angepasstes Aussehen oder andere Markup-Sprachen sind möglich.
- ▼ Die Kommunikation zwischen GUI-Elementen und verarbeitenden Funktionen findet über ein Event/Listener Konzept statt.
- ▼ Jeder GUI-Komponente kann ein Validator zugeordnet werden, der das Format eingegebener Daten prüft.
- ▼ Dialogabläufe werden in der Datei `faces-config.xml` spezifiziert.

Grobarchitektur von JavaServer Faces

Abbildung 2 stellt den prinzipiellen Aufbau einer JSF-Anwendung dar. Einige JSF-Elemente wurden zur Vereinfachung weggelassen. Die orangefarbenen und grünen Elemente sind vom Entwickler zu implementieren bzw. zu spezifizieren. Die blauen Elemente gehören zu JSF oder werden zur Laufzeit von den JSF erzeugt.

Die `faces-config.xml`-Datei konfiguriert die Anwendung. Sie definiert beispielsweise die verwendeten JavaBeans und die Dialogsteuerung (s. u.).

Das `FacesServlet` nimmt die Requests vom Browser entgegen und leitet sie an die JSPs weiter. Die JSPs verwenden die JSF-Tag Library und bauen darüber für jede JSP-Seite einen eigenen GUI-Komponentenbaum auf.

Die Daten aus dem Request werden auf den Komponentenbaum abgebildet. Die Daten aus dem Komponentenbaum werden ihrerseits auf JavaBeans abgebildet. Werden in der Oberfläche Buttons (oder vergleichbare Elemente) betätigt, werden über einen Event-Mechanismus die ActionListener aufgerufen, die an diesem Oberflächen-element angemeldet sind.

Die JSF-Spezifikation beschreibt die Verarbeitung eines Requests mit dem *Standard Request Processing Lifecycle*. Dieser teilt die Verarbeitung eines Requests und die Erzeugung des Ergebnisses in sechs Phasen ein.

Abbildung 3 ist vereinfacht aus der Spezifikation übernommen, die Fehlerverarbeitung wurde aus Gründen der Übersichtlichkeit weggelassen.

Die Phasen sind mit ihren Verantwortlichkeiten in Tabelle 1 dargestellt. Zaunick [Zau04] bietet eine detaillierte Darstellung. Nach fast jeder Phase werden eventuell angefallene Events verarbeitet. Fehlgeschlagene Validierungen und Events erzeugen automatisch eine Fehlerseite.

Phase	Verantwortlichkeit
Restore View	Komponentenbaum wird erzeugt bzw. wieder hergestellt, Validatoren und Event-Handler werden an die GUI-Komponenten gekoppelt
Apply Request Values	Daten aus dem Request werden in den Komponentenbaum übertragen
Process Validations	Eventuell angemeldete Validatoren prüfen die Daten aus den GUI-Komponenten. Fehler werden gemeldet
Update Model Values	Daten werden aus den GUI-Komponenten in die zugeordneten JavaBeans kopiert
Invoke Application	Aufrufen der Geschäftslogik und Navigation zwischen JSP-Seiten
Render Response	Die Ergebnisseite wird erzeugt, dafür werden die Renderer verwendet

Tabelle 1: Phasen des JSF Request Processing Lifecycle

GUI-Komponenten

Oberflächen werden über JSPs definiert und durch einen Komponentenbaum aus `UIComponent`s repräsentiert. `UIComponent` ist das Basisinterface für alle GUI-Komponenten. Implementierungen sind beispielsweise `UICommand` (für Buttons, Hyperlinks etc.), `UIOutput` oder `UIInput` (für Eingabefelder etc.). Spezielle Implementierungen (z. B. für spezifische Seitenlayouts [Zau04]) können leicht ergänzt werden.

Die GUI-Komponenten in JSF haben folgende Aufgaben:

- ▼ Darstellen der Komponente in HTML oder einer anderen Markup-Sprache, in der Regel über einen Renderer,
- ▼ Verarbeiten der Events, welche an die Komponente gesendet werden,
- ▼ Validieren der Benutzereingaben,
- ▼ Verwalten der eigenen Attribute.

Renderer

Jede `UIComponent` kann entweder sich selber rendern, dafür sind im Interface Methoden vorgesehen, oder über die Methode `getRendererType()` einen zuständigen Renderer benennen. Der Renderer dekodiert die Daten aus dem Request. Bei der Erstellung der Ergebnis-Seite erzeugt er die Darstellung des GUI-Elements in HTML oder einer anderen Markup-Sprache.

Validatoren

In der JSP-Seite kann jeder GUI-Komponente ein Validator zugeordnet werden. Dieser prüft die Nutzereingaben und erzeugt ggf. eine Fehlermeldung. Die Position der Fehlermeldung auf der Seite wird durch ein `<h:message/>`-Tag bestimmt.

Validatoren werden derzeit für verschiedene Datentypen angeboten, das sind unter anderem: der `LengthValidator`, welcher String-Längen prüft,

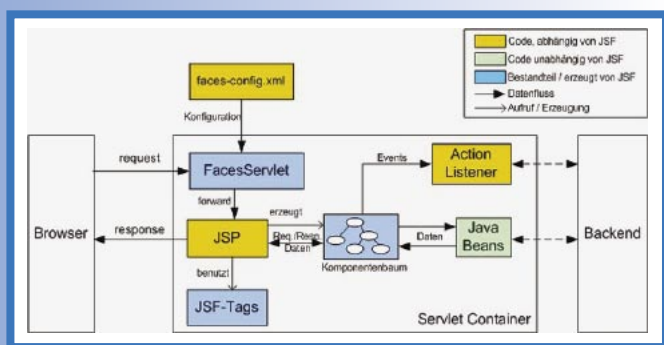


Abb. 2: Grober Aufbau einer JSF-Anwendung

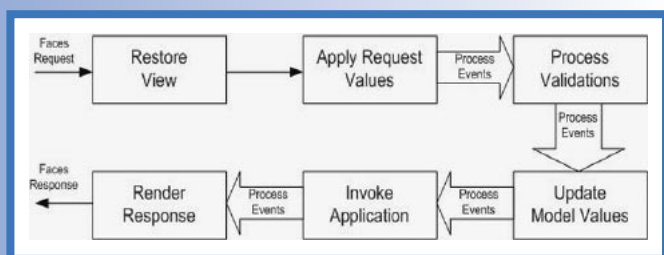


Abb. 3: JSF Request Processing Lifecycle

oder der `DoubleRangeValidator`, welcher einen Zahlenwert gegen Minimal- und Maximalwert prüft. Zusätzlich besteht die Möglichkeit, eigene Validatoren zu implementieren. Der nachfolgende JSP-Ausschnitt meldet einen `LengthValidator` an dem Eingabefeld für den Vornamen aus unserem ersten Beispiel an. Dieser erzeugt eine Fehlermeldung, wenn der eingegebene Vorname weniger als 2 oder mehr als 20 Zeichen hat:

```
<h:input_text id="vorname" value="#{user.vorname}">
  <f:validate_length maximum="20" minimum="2"/>
</h:input_text>
```

Alternativ besteht die Möglichkeit, direkt eine Methode der zugrunde liegenden Bean anzugeben, um eine Validierung durchzuführen:

```
<h:input_text id="nachname" value="#{user.nachname}"
  validator="#{user.validate}" />
```

Events und Listener

Wie bei Swing bieten auch die JavaServer Faces ein Event-Konzept an. An Oberflächenelementen werden Listener angemeldet. Die Elemente erzeugen Events, die dann von den Listnern verarbeitet werden. Es gibt zwei Arten von Events: Wenn ein Button oder ein anderes `UICommand` gedrückt wird, wird ein `ActionEvent` ausgelöst. Ein `ValueChangedEvent` wird ausgelöst, wenn sich der Wert in einem Texteingabefeld oder einem anderen `UIInput` geändert hat.

Listener verarbeiten Events: `ActionListener` verarbeiten `ActionEvents` und `ValueChangedListener` verarbeiten `ValueChangedEvents`. Der folgende Ausschnitt aus der JSP von oben registriert den Listener `WelcomeActionListener` an dem `WELCOME`-Button:

```
<h:command_button id="welcome" action="success" value="WELCOME">
  <f:action_listener type="WelcomeActionListener"/>
</h:command_button>
```

Wird der `WELCOME`-Button betätigt, wird am Server ein entsprechendes `ActionEvent` ausgelöst. Dieses enthält Informationen über seine Herkunft. Der nachfolgende Quelltext zeigt die Implementierung des `WelcomeActionListeners`. Die Methode `processAction` verarbeitet das Event:

```
public class WelcomeActionListener implements ActionListener {
  public WelcomeActionListener() {}

  public void processAction(ActionEvent event) {
    String id = event.getComponent().getId();
    FacesContext context = FacesContext.getCurrentInstance();
    if (id.equals("welcome")) {...}
    // Auslesen weiterer Informationen über context
  }
}
```

Der Listener kann durchaus bei mehreren Komponenten der Oberfläche angemeldet werden. Die verursachende Komponente ist über die `getComponent()`-Methode des Events zu erfragen.

Datenaustausch

Der Lebenszyklus der verwendeten JavaBeans wird durch das JSF-Framework verwaltet. Das Erzeugen von Beans über das `<jsp:useBean ... >`-Tag ist nicht mehr erforderlich. In der `faces-config.xml`-Datei

werden die verwalteten JavaBeans aufgelistet. Ihre Lebensdauer ist ebenfalls anzugeben. Der folgende Ausschnitt meldet `UserBean` zur Verwaltung an. Der Scope gibt an, dass für jeden Request ein neues `UserBean`-Objekt erzeugt wird. `None`, `Session` und `Application` sind weitere mögliche Scopes.

```
<managed-bean>
  <description />
  <managed-bean-name>user</managed-bean-name>
  <managed-bean-class>UserBean</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Über den in der `faces-config`-Datei angegebenen Namen wird auf das `UserBean` zugegriffen. Das folgende Eingabefeld liest bzw. schreibt beispielsweise das `vorname`-Property der Bean. Die Verbindung zwischen `JavaBean` und `GUI-Element` wird über das `value`-Attribut hergestellt. Hierbei muss die im Beispiel gezeigte „#{...}“-Syntax verwendet werden, da auch die Möglichkeit besteht an dieser Stelle ein Literal zu verwenden.

```
<h:input_text id="vorname" value="#{user.vorname}"/>
```

Dialogabläufe

Dialogabläufe werden häufig nur implizit im Code spezifiziert. JavaServer Faces definieren diese explizit in der `faces-config.xml`-Datei. Im Prinzip wird ein gerichteter Graph mit JSPs als Knoten und Aktionen als Kantenmarkierungen definiert. Das folgende Listing definiert für die `welcome.jsp` die zwei Nachfolgeseiten `application.jsp` und `problem.jsp`.

```
<navigation-rule>
  <from-view-id>/welcome.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/application.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>failure</from-outcome>
    <to-view-id>/problem.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Das `<from-outcome ... >`, also eine Kantenmarkierung in dem gerichteten Graphen, kann einem `<command_button>` als Action-Attribut zugeordnet werden. Damit ist jedoch die Navigation fest in der JSF-Seite codiert, im Beispiel wird die Seite `problem.jsp` niemals erreicht.

```
<h:command_button action="success" value="WELCOME"/>
```

Um die Wahl des Navigationspfads explizit steuern zu können, muss eine Action-Methode implementiert und vom `<command_button>` mit der bereits bekannten „#{...}“-Syntax referenziert werden:

```
<h:command_button action="#{WelcomeServer.logonAction}"
  value="WELCOME"/>
```

Bei der Action-Methode handelt es sich um eine parameterlose Methode, die einen String zurück liefert. Ihr Rückgabewert legt mit dem Navigationsgraphen die JSP fest, welche nach `welcome.jsp` angezeigt wird.



```
public class WelcomeServer {
    public String logonAction() {
        boolean success = ...;
        if (success) return "success";
        return "failure";
    }
}
```

Die Action-Methode ist neben den Event/Listenern (s. o.) der zweite Weg, Geschäftslogik an die JSF-Seiten anzubinden.

Fazit

Die JavaServer Faces-Spezifikation enthält viele interessante Elemente, welche an Java-Swing erinnern: serverseitig wird ein GUI-Komponentenbaum erzeugt, der für die Darstellung der Seiten verantwortlich ist. Verarbeitender Code wird über ein Event-Listener-Konzept angebunden.

Leider wird das viel versprechende GUI-Komponentenmodell über Tag-Libs dargestellt und nicht über Java-Code. Die vorgesehene Art der Verwendung der JSF führt schnell zum bekannt unübersichtlichen HTML-Java Mix. Die Verteilung der Oberfläche auf HTML/JSF-Seiten auf der einen Seite und auf dahinter stehende Java-Klassen auf der anderen Seite macht die Technologie leider ungeeignet für die aktuellen Themen Refactoring und Test Driven Development; zumindest so lange keine Werkzeuge verfügbar sind, die beide Seiten gemeinsam unterstützen.

Andere Lösungen, wie beispielsweise das W4Toolkit [W4T03] der Karlsruher Firma Innoopract, erlauben dagegen eine Swing-ähnliche Web-Programmierung und setzen dabei konsequent auf Java.

Nichtsdestotrotz stellen die JSF eine interessante Technologie dar, die, wie das allgemeine Medieninteresse zeigt, einige Bewegung in den Web-Framework-Markt bringen dürften.

Literatur und Links

- [EA4] JavaServer Faces, Sun Early Access Version 4, <http://java.sun.com/j2ee/javaxserverfaces/>
- [JSF03] JavaServer Faces Specification 1.0, Proposed Final Draft, <http://java.sun.com/j2ee/javaxserverfaces/>
- [Str03] Struts Online Manual, <http://jakarta.apache.org/struts/>
- [W4T03] W4 Toolkit-Website, <http://www.w4toolkit.com>
- [Zau04] K. Zaunick, Gut zu Gesicht: Java Server Faces, JavaSPEKTRUM, 1/2004



Gerd Beneken ist wissenschaftlicher Angestellter am Lehrstuhl für Software und Systems Engineering der TU-München. Davor war er Technischer Berater bei der sd&m AG. Er verfügt über langjährige Erfahrungen in Entwicklung und Design großer verteilter objektorientierter Systeme. Außerdem ist er Lehrbeauftragter für Software Engineering an der FH Rosenheim.
E-Mail: gerd.beneken@in.tum.de.

Florian Deissenböck studierte Informatik an der TU München und am Asian Institute of Technology, Bangkok mit Schwerpunkten Software Engineering, Benutzeroberflächen und Compilerbau. Er ist wissenschaftlicher Angestellter am Lehrstuhl für Software und Systems Engineering der TU-München.
E-Mail: florian.deissenboeck@in.tum.de.