

JSF Grundlagen

Michael Müller

JSF Grundlagen

Die Oberfläche von Java / Jakarta EE

JSF Grundlagen

Diese Foliensammlung stellt stichpunktartig die Schulungsinhalte dar.

Sie dient lediglich als Leitfaden und ist insbesondere kein Ersatz für die Schulung oder ein Lehrbuch.

Im Rahmen der Schulung werden die Inhalte ausführlich erläutert und durch praktische Übungen vertieft.

Der Referent

- Michael Müller
- Mehr als 30 Jahre Softwareentwicklung
- Bereichsleiter Softwareentwicklung
- Freier Autor für diverse Fachzeitschriften / Onlinemedien
- Freier Buch-Autor (vornehmlich zu Java, JSF)
- Mitglied in der JSF Expert Group (JSR 344, JSR 372)
- Unregelmäßiger Blogger: blog.mueller-bruehl.de

Enterprise GUI

- Verteilte Anwendung
- Nutzung auf beliebigem Betriebssystem
- Web-Anwendung mit Darstellung im Browser
- Warum JSF, wo es doch andere Sprachen bzw. Webframeworks gibt?

Good old JSF?

- *Setzen Sie tendenziell immer auf die langweiligste und am besten etablierte Technologie, die zur Umsetzung der jeweiligen Anforderungen ausreichend ist. Etablierte Technologien haben diverse Vorteile [...]. Und wenn es schon länger existiert, stehen die Chancen gut, dass es auch weiterhin Support dafür geben wird. Viele „Shiny New Objects“ aus dem Internet, wie neue Frameworks zur Entwicklung von Web-GUIs, haben teilweise eine kürzere Haltbarkeitsdauer als die meisten Dinge aus dem nächsten Supermarkt.*

(Herbert Dowalil, Grundlagen des modularen Softwareentwurfs)

PHP (Login wordpress)

```
<!DOCTYPE html>
<!--[if IE 8]>
<html xmlns="http://www.w3.org/1999/xhtml" class="ie8" <?php language_attributes(); ?>>
<![endif]-->
<!--[if !(IE 8) ]><!-->
<html xmlns="http://www.w3.org/1999/xhtml" <?php language_attributes(); ?>>
<!--<![endif]-->
<head>
<meta http-equiv="Content-Type" content="<?php bloginfo('html_type'); ?>; charset=<?php bloginfo('charset'); ?>" />
<title><?php echo $login_title; ?></title>
<?php
wp_enqueue_style( 'login' );
... weiterer PHP code ...
```

- In HTML eingestreuter Code
 - Für HTML-Natives leichter zu erlernen, da keine zusätzliche Abstraktion
 - Jedoch schlechter zwischen Code und Inhalt zu trennen
 - Moderne Frameworks erlauben bessere Trennung zwischen Code und Layout
- Serverseitige Ausführung und Validierung
 - Die Applikation läuft in kontrollierter Umgebung

- Dynamische Sprache
 - Keine Typprüfung durch Compiler; falsche Typen fallen oftmals erst zur Laufzeit auf
- Kein Java
 - Damit für JEE nicht geeignet

Angular

```
<body ng-app="realestateApp" class="ng-scope">
  <div class="panel panel-default">
    <div class="panel-heading">
      Real Estate App
    </div>
    <div class="container" ng-controller="tenantsController" ngcloak>
      <h1>Tenants</h1>
      <div class="row">
        <div>
          <button class="btn btn-primary" ng-click="previousTenant()">
            Previous
          </button>
        </div>
      </div>
    </div>
  </div>
</body>
```

Angular

- Angular als Beispiel für beliebige clientseitige JavaScript Frameworks
- Spezielle Attribute im HTML
- Ausführung auf dem Client
 - Eingabevalidierung ohne Serverzugriff möglich
 - Erfordert hohe Disziplin bei der Programmierung, da die Eingabevalidierung serverseitig nicht vergessen werden darf!

Angular

- Dynamische Sprache
 - Keine Typprüfung durch Compiler; falsche Typen fallen oftmals erst zur Laufzeit auf
- Kommuniziert mit Server häufig via REST
- Für JEE geeignet
 - Da das clientseitige Programm mit dem Server via HTTP / REST kommuniziert, kann die serverseitige Applikation beliebig implementiert werden
 - JEE kann beispielsweise via Jax-RS angebunden werden

Java Server Faces

- Spezielle Attribute bzw. Namespaces im HTML
 - Bei Verwendung des HTML friendly Markup
- Ausführung auf dem Server
 - Einabevalidierung erfordert Serverzugriff
 - Damit sind die serverseitigen Daten automatisch geprüft; ein Bypassing via Curl etc. ist nicht möglich

Java Server Faces

```
<head>
  <title>TinyCalculator</title>
</head>
<body>
  <h1>TinyCalculator</h1>
  <form jsf:id="calc">
    <div>
      Param1: <input type="text" jsf:value="#{tinyCalculator.param1}"/>
    </div>
    <div>
      Param2: <input type="text" jsf:value="#{tinyCalculator.param2}"/>
    </div>
    <div>
      <input type="submit" value="Add"
        jsf:action="#{tinyCalculator.add}"/>
    </div>
  </form>
</body>
```

Java Server Faces

```
<h:head>
  <title>TinyCalculator</title>
</h:head>
<h:body>
  <h1>TinyCalculator</h1>
  <h:form id="form">
    <div>
      <h:outputLabel value="Param1: "/>
      <h:inputText id="param1" value="#{tinyCalculator.param1}"/>
      <h:message for="param1" errorClass="error"/>
    </div>
    <div>
      <h:outputLabel value="Param2: "/>
      <h:inputText id="param2" value="#{tinyCalculator.param2}"/>
    </div>
    <div>
      <h:commandButton id="add" value="Add" action="#{tinyCalculator.add}"/>
    </div>
  </h:form>
</h:body>
```

Java Server Faces

- JSF Tags für Komponenten
- Beliebige Mischformen möglich
 - HTML friendly und / oder JSF Tags
- Definition eigener (composite) Komponenten via Tags
- Nutzung vorgefertigter Komponenten

Java Server Faces

- Trennung von Content/Layout und Programmierung
 - Erlaubt arbeitsteilige Entwicklung durch Web-Designer und Programmierer
 - Die zusätzliche Abstraktion ist für Einsteiger, die beide Rollen ausfüllen manchmal schwieriger
 - JSF Tags erlauben es, Seiten weitgehend ohne HTML-Kenntnisse zu entwickeln
 - Vadin geht hier noch einen Schritt weiter, doch welcher Entwickler verfügt heute nicht über HTML-Wissen?

Java Server Faces

- Expression Language (EL) als Bindeglied zum Source
- EL erlaubt den Zugriff auf CDI-Beans
 - Zugriff auf die veralteten JSF managed Beans erfolgt in gleicher Art und Weise, jedoch in neuen Projekten nicht mehr nutzen!
- Direkte Ausnutzung der JEE Features

JSF Eigenschaften

- User Interface Framework für Webanwendungen
- Besonders geeignet für datengetriebene Anwendungen
 - Insbesondere formularlastige Anwendungen
- Frontend für Enterpriseanwendungen

JSF Eigenschaften

- Handhabung Komponentenstatus über mehrere Requests hinweg (HTTP)
- Formularverarbeitung, auch über mehrere Seiten
- Streng typisiertes serverseitiges Evenmodel für die vom GUI erzeugten Ereignisse
- Seitennavigation als Antwort auf bestimmte Ereignisse

Die letzte JavaServer Faces Release-Party*

- Java EE 8
Release 09/2017
- JSF 2.3
Release 03/2017
- Releaseparty im
JavaLand 2017
- *Künftig wohl Jakarta Server Faces

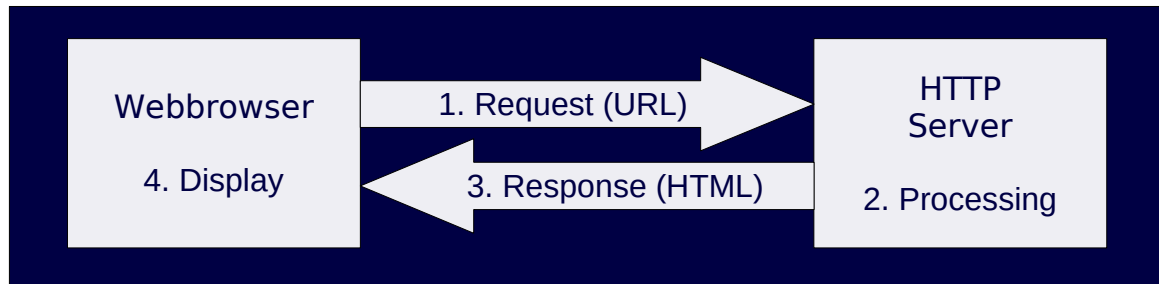


Exkurs: Telefonverbindung

- Teilnehmer wählt Zielnummer
- Angerufener Teilnehmer hebt ab
 - Verbindung zwischen zwei Endpunkten aufgebaut
- Bidirektionale Verbindung
 - Beide Teilnehmer können jederzeit sprechen und hören (zumindest auf technischer Ebene)
- Auflegen beendet

HTTP

- Zustandslos
- Server lauscht auf Anfrage, versendet Antwort und kappt die Verbindung



HTTP

- Problem HTTP: Zustandslos
- JavaServer Faces übernimmt die Sessionverwaltung
- Zuordnung der Anwendersitzung zu einem komplexen serverseitigen Ablauf
- Eine Session muss nicht notwendigerweise mit der gesamten Nutzungsdauer im Browser übereinstimmen

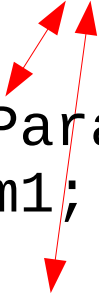
z.B. Generierung einer neuen Session aus Sicherheitsgründen bei An- oder Abmeldung

HTML-Seite und Model

■ Nutzung der Expression Language

```
<h:inputText  
    id="param1"  
    value="#{calculatorBean.param1}"/>
```

```
public int getParam1() {  
    return _param1;  
}  
public void setParam1(int param1) {  
    _param1 = param1;  
}
```

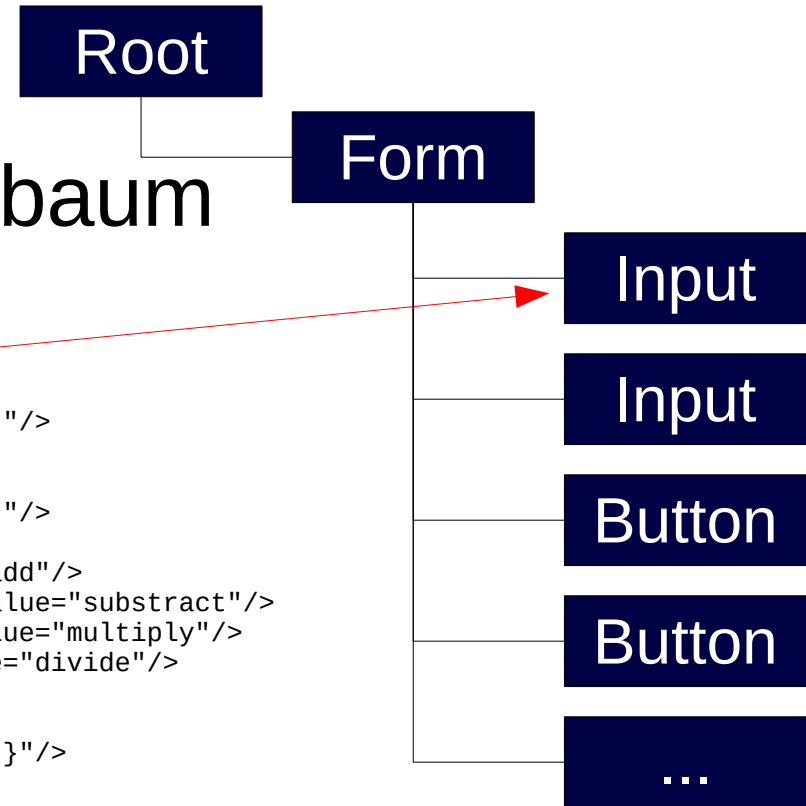


The diagram consists of three red arrows. One arrow points from the `setParam1` method to the `getParam1` method. Two other arrows point from the `getParam1` method to the `value` attribute of the `<h:inputText>` tag in the code block above.

Komponentenbaum

■ HTML Seite und stark vereinfachter Komponentenbaum

```
[...]
<h:body>
  <h1>Tiny calculator</h1>
  <h:form id="calculator">
    <h:outputLabel for="param1" value="Param 1:"/>
    <h:inputText id="param1" value="#{calculatorBean.param1}"/>
    <br/>
    <h:outputLabel for="param2" value="Param 2:"/>
    <h:inputText id="param2" value="#{calculatorBean.param2}"/>
    <br/>
    <h:commandButton action="#{calculatorBean.add}" value="add"/>
    <h:commandButton action="#{calculatorBean.subtract}" value="subtract"/>
    <h:commandButton action="#{calculatorBean.multiply}" value="multiply"/>
    <h:commandButton action="#{calculatorBean.divide}" value="divide"/>
    <br/>
    <h:outputLabel for="result" value="Result:"/>
    <h:outputText id="result" value="#{calculatorBean.result}"/>
  </h:form>
</h:body>
[...]
```



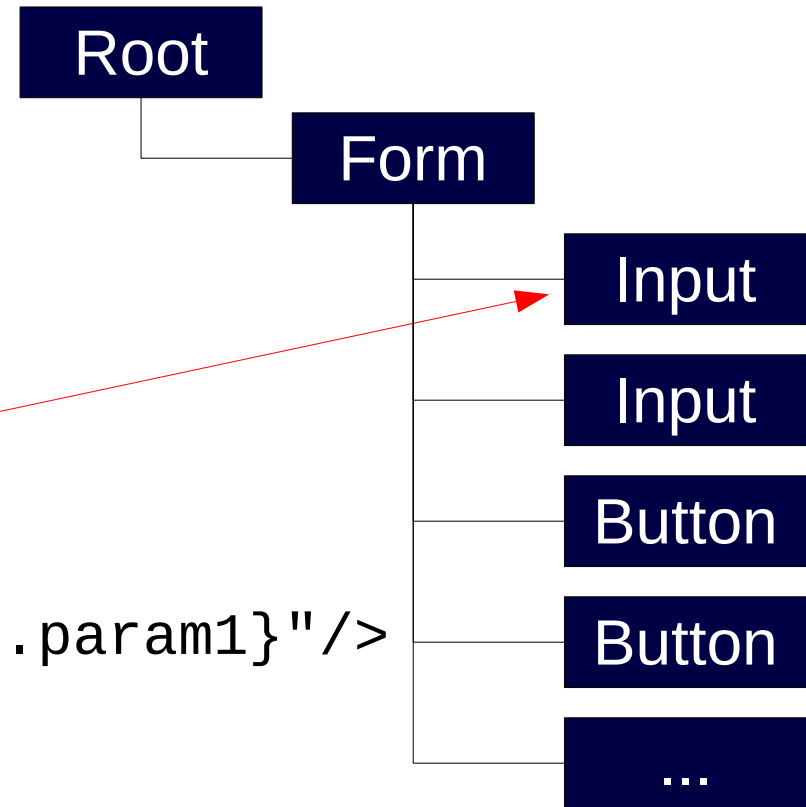
Komponentenbaum

- Hier der Zoom:

[...]

```
<h:inputText  
id="param1"  
value="#{calculatorBean.param1}"/>
```

[...]



Eclipse IDE

- Eclipse IDE for Enterprise Java Developers
 - Download von
<https://www.eclipse.org/downloads/packages/>
- Entpacken in beliebiges Verzeichnis
- Start

Eclipse IDE

- Unterstützung für WildFly
 - Help, Marketplace, Update Marketplace Client
 - Suche nach WildFly und Installation JBoss Tools
 - In *Servers* view neuen Server erstellen
 - Auswahl Red Hat JBoss Middleware, Download erfolgt automatisch
 - Nochmals neuen Server erstellen und nun WildFly 16 wählen: Download und Install

Eclipse IDE

- *Window, Preferences, JBoss Tools, Web, Editors, Visual Page Editor, Code Templates:*
- In der Namespacedefinition *java.sun.com* durch *xmlns.jcp.org* ersetzen.

Navigation

- Einfach als Rückgabewert einer Aktion
- Null oder Leerstring verbleibt auf der Seite
- String gibt Ziel an

```
public String add(){  
    [...]  
    return "";  
}
```

```
public String add(){  
    [...]  
    return "result.xhtml";  
}
```

- Zielseite dynamisch bestimmbar
- Navigation via Konfiguration (XML)

Navigation

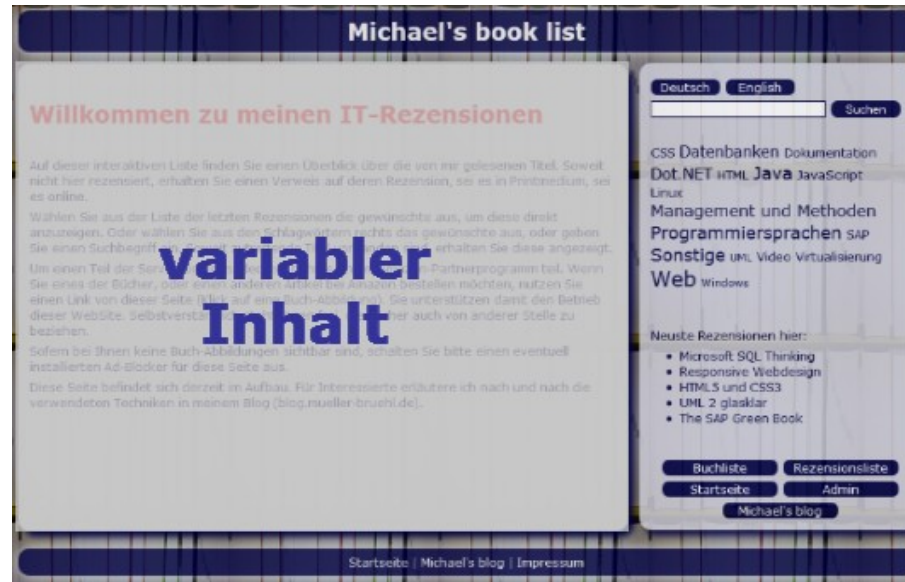
- Navigation per „Text“ fehleranfällig
- Navigation mit XML umständlich
- Einsatz einer zentralen Page-Enum
- Mit JSF 2.3 direkt nutzbar
- Bis JSF 2.2 indirekt via Bean nutzbar

Navigation

- Postback-Navigation
 - Serveraufruf mittels HTTP Post, bedingt durch HTML-Formular
- Get-Navigation
 - Bookmarkable, sofern erforderlich
- Postback und Redirect

Templates

- Templates sind Schablonen
- „Fester“ Inhalt mit Fenster zu variablen Teilen



Templates

■ Auszug aus einem Template

```
[...]
<div id="top">
    <h1>Michael's book list</h1>
</div>
<div id="main">
    <ui:insert name="content">Content</ui:insert>
</div>
<div id="sidebar">
    <h:form id="sidebarform">
[...]
```

Templates

■ Nutzung des Templates

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE ...>
<ui:composition
  xmlns:ui="http://java.sun.com/jsf/facelets"
  template="booksTemplate.xhtml" ...>

  <ui:define name="content">
    [...seitenspezifischer Inhalt...]
  </ui:define>
</ui:composition>
```

Scopes

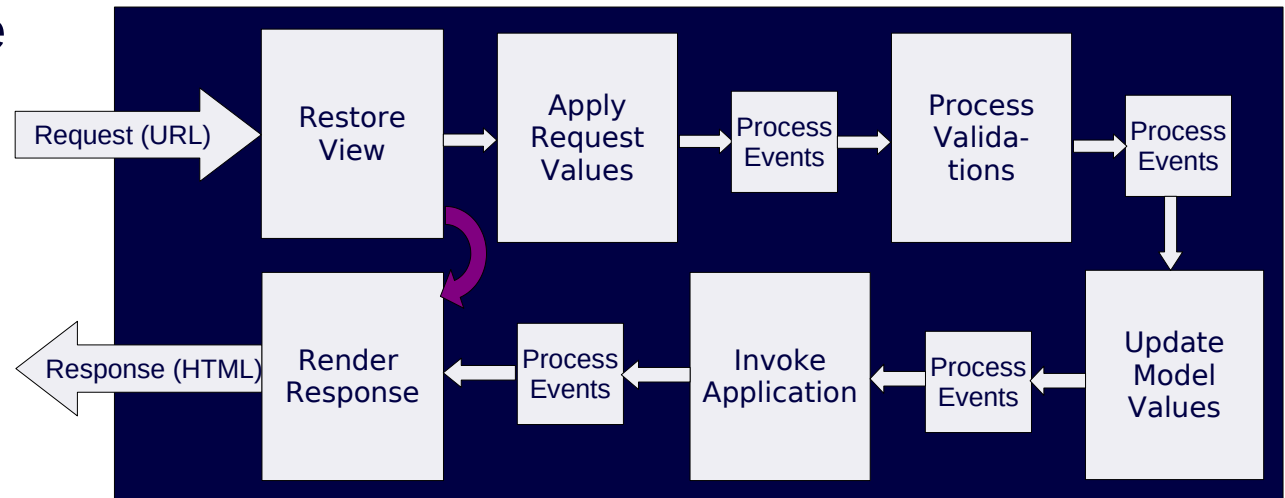
- CDI/JSF bieten mit „Scopes“ einen Mechanismus, die Lebensdauer eines Objekts zu bestimmen
- CDI: Request, Conversation, Session, Application, Dependent
- JSF, basierend auf CDI: View, Flow

AJAX

- Partial Rendering mittels Tag zufügen
- `<f:ajax .../>`

JSF Lebenszyklus

- Initialer Request: Restore View, Render Resp.
- Folge request: Fluss durch alle Phasen
- Abweichungen in Process Events möglich
 - Response Complete (End Lifecycle),
 - Render Response



Validation

- Automatisch, abhängig vom Datentyp
- Validator-Methode
- Validator-Tag
- Bean-Validation
- Multi-Field-Validation (ab JSF 2.3)

Komponenten

- Composite Component (seit JSF 2.0)
- Definition mit HTML/Tags wie eine Seite
- Definition von Interface und Implementierung
- Beispiel: `labeledText`

Loop

- Schleifen mittels `<ui:repeat>`
- Tabellen mittels `<h:dataTable>`
- Schleifen mittels `<c:forEach>` (JSP)

Converter

- Im Browser reine Stringdarstellung
- Objekte nach String sowie zurück konvertierten

```
@FacesConverter(value = "ProductConverter")
public class ProductConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext ctx, UIComponent component, String value) {
        [...]
    }

    @Override
    public String getAsString(FacesContext ctx, UIComponent component, Object value) {
        [...]
    }
}
```

Ressourcen

■ Im Verzeichnis resources

```
<h:outputStylesheet library="#{sessionTools.theme}" name="css/demo.css" />
```

```
<h:outputScript library="#{sessionTools.theme}" name="script/demo.js" />
```

Message Bundle

■ Definition in faces-config.xml

```
<message-bundle>de.muellerbruehl.demo.messages</message-bundle>
<locale-config>
  <default-locale>de</default-locale>
  <supported-locale>en</supported-locale>
</locale-config>
<resource-bundle>
  <base-name>de.muellerbruehl.demo..messages</base-name>
  <var>msg</var>
</resource-bundle>
```

Komponentenframeworks

- **Apache Tobago**

<https://myfaces.apache.org/tobago/index.html>

- **ICEfaces**

<http://www.icesoft.org/java/projects/ICEfaces/overview.jsf>

- **PrimeFaces**

<https://www.primefaces.org/#primefaces>

- **Andere Frameworks sind „End of Life“**

z.B. RichFaces, Trinidad etc.

Abschluss

Fragen?

**Danke für Eure
Aufmerksamkeit!**