2.– 5. September 2013
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Neues Gesicht

Neues und Erwähnenswertes aus JavaServer Faces 2.2

Michael Kurz

JavaServer Faces im Rückblick

Big Ticket Features

- HTML5 Friendly Markup
- Resource Library Contracts
- Faces Flows
- Stateless Views

HTML5

- Neue Elemente
- Neue Eingabetypen
- Neue Attribute
- Custom-Data-Attribute

HTML5: Klassisches JSF

- Seite mit JSF-Tags

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head><title>JSF 2.2 HTML5</title></h:head>
<h:body>
  <h:form id="form">
    <h:inputText id="email" value="#{bean.email}"/>
    <h:commandButton action="#{bean.save}"
        value="Save"/>
  </h:form>
</h:body>
</html>
```

HTML5: Pass-Through-Attribute

- Seite mit JSF-Tags + HTML5-Attributen

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:pt="http://xmlns.jcp.org/jsf/passthrough">
<h:head><title>JSF 2.2 HTML5</title></h:head>
<h:body>
  <h:form id="form">
    <h:inputText id="email" value="#{bean.email}"
        pt:type="email" pt:placeholder="Ihre Email"/>
    <h:commandButton action="#{bean.save}"
        value="Save"/>
  </h:form>
</h:body>
</html>
```

HTML5: Pass-Through-Attribute

- Pass-Through-Attribute über Tags

```
<h:inputText id="email" value="#{bean.email}">
  <f:passThroughAttribute name="type" value="email"/>
  <f:passThroughAttribute name="placeholder"
      value="Ihre Email"/>
  <f:passthroughAttributes value="#{bean.attrs}"/>
</h:inputText>
```

- HTML5-Custom-Data-Attributes

```
<h:inputText id="email" value="#{bean.email}"
    pt:type="email" pt:data-required="true"/>
```

HTML5: Pass-Through-Elemente

- Seite mit HTML5 + JSF-Attributen

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:jsf="http://xmlns.jcp.org/jsf">
<head jsf:id="head"><title>JSF HTML5</title></head>
<body jsf:id="body">
  <form jsf:id="form">
    <input jsf:id="name" jsf:value="#{bean.name}"
        type="text" placeholder="Enter name"/>
    <button jsf:action="#{bean.save}">Save</button>
  </form>
</body>
</html>
```

HTML5: Mapping zu JSF-Tags

HTML-Element	Selektor-Attribut	JSF-Tag
a	jsf:action	h:commandLink
a	jsf:outcome	h:commandLink
form		h:form
input	type="button"	h:commandButton
input	type="checkbox"	h:selectBooleanCheckbox
input	type="file"	h:inputFile
input	type="*"	h:inputText
select	multiple="*"	h:selectManyListbox
select		h:selectOneListbox

HTML5: Pass-Through-Elemente

- Pass-Through-Elemente werden Komponenten

```
<input type="email" jsf:value="#{bean.name}">
  <f:validateLength minimum="3"/>
</input>
```

- Generische Pass-Through-Elemente

```
<progress jsf:id="progress" max="10"
    value="#{bean.progress}">
  <f:ajax event="click" render="@this"/>
</progress>
```

Demonstration

HTML5 Friendly Markup

- Pass-Through-Attribute
- Pass-Through-Elemente

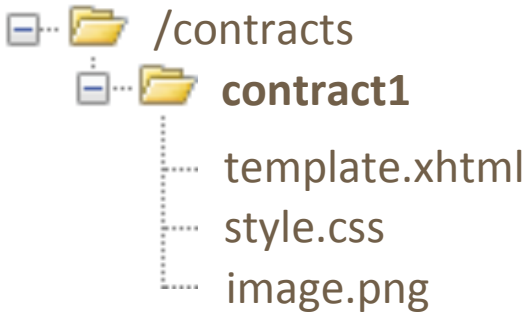
Beispiel: **https://github.com/jsflive/jsf22-examples/jsf22-html5**

Resource Library Contracts

- Resource Library Contract
  - Templates
  - Ersetzbare Bereiche
  - Ressourcen

Contracts: Ein erstes Beispiel 1/3

- JSF sucht Contracts an folgenden Stellen
  - **/contracts**
  - **/META-INF/contracts**
- Beispiel **contract1**



Contracts: Ein erstes Beispiel 2/3

- Template **template.xhtml**

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head><title>Contract Template</title></h:head>
<h:body>
  <h:outputStylesheet name="style.css"/>
  <div class="header">
    <ui:insert name="header"/>
  </div>
  <div class="content">
    <ui:insert name="content"/>
  </div>
</h:body>
</html>
```

Contracts: Ein erstes Beispiel 3/3

- Template-Client

```
<ui:composition template="/template.xhtml">
  <ui:define name="header">
    <h1>Überschrift</h1>
  </ui:define>
  <ui:define name="content">
    <p>Beliebiger Inhalt</p>
    <h:graphicImage name="image.png"/>
  </ui:define>
</ui:composition>
```

Contracts: Zuordnung

- Standard: Alle Seiten bekommen alle Contracts
- Statische Zuordnung in **faces-config.xml**

```
<resource-library-contracts>
  <contract-mapping>
    <url-pattern>/special/*</url-pattern>
    <contracts>layout-special</contracts>
  </contract-mapping>
</resource-library-contracts>
```

- Dynamische Zuordnung pro Seite

```
<f:view contracts="contract1">
  <ui:composition template="/template.xhtml">
    ...
```

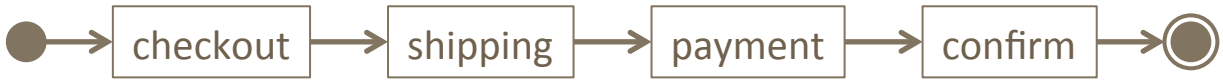
Demonstration

Resource Library Contracts

Beispiel: **https://github.com/jsflive/jsf22-examples/jsf22-contracts**

Faces Flows

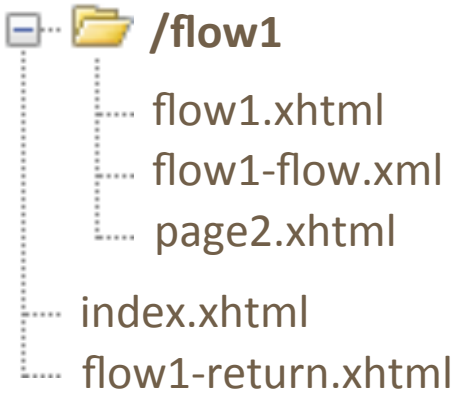
- Flow gruppiert Seiten
- Definierte Start- und Endpunkte



- Flows ermöglichen:
  - Wiederverwendbarkeit
  - Navigation auf Flow einschränken
  - Flow-Scope

Flows: Implizite Flow-Definition

- Definition in Verzeichnis **/<flow-ID>**
- Leere Konfiguration: **<flow-ID>-flow.xml**
- Startknoten: **<flow-ID>.xhtml**
- Endknoten: **/<flow-ID>-return.xhtml**



Flows: Impliziten Flow verwenden

- Flow starten:

```
<h:link value="Flow 1" outcome="flow1"/>
```

- Navigation innerhalb Flow:

```
<h:commandButton value="Next" action="page2"/>
```

- Flow beenden:

```
<h:button value="Finish" outcome="flow1-return"/>
```

Flows: Explizite Flow-Definition (XML)

- Konfiguration in **<flow-ID>-flow.xml**

```
<faces-config version="2.2">
  <flow-definition id="flow1">
    <flow-return id="flow1-return">
      <from-outcome>/index.xhtml</from-outcome>
    </flow-return>
  </flow-definition>
</faces-config>
```

Flows: Explizite Flow-Definition (Java)

- Konfiguration mit CDI-Producer

```
public class Flow1 implements Serializable {

  @Produces @FlowDefinition
  public Flow buildFlow(@FlowBuilderParameter
      FlowBuilder builder) {
    String flowId = "flow1";
    builder.id("", flowId);
    builder.viewNode("flow1", "/flow1/flow1.xhtml")
           .markAsStartNode();
    builder.returnNode("flow1-return")
           .fromOutcome("/index.xhtml");
    return builder.getFlow();
  }
}
```

Flows: Knotentypen

- View Node
- Return Node
- Flow Call Node
- Switch Node
- Method Call Node

Flows: @FlowScoped

- CDI-Scope für einen Flow

```
@javax.inject.Named
@javax.faces.flow.FlowScoped(value="flow1")
public class FlowBean {
   ...
}
```

- An Browsertab/-fenster gebunden
- Implizites Objekt **flowScope** (Map)

```
<h:outputText value="#{flowScope.param1}"/>
```

Demonstration

Faces Flows

Beispiel: **https://github.com/jsflive/jsf22-examples/jsf22-flows**

Stateless Views

- State für Seite nicht speichern

```
<f:view transient="true">
  ...
</f:view>
```

Neue Namensräume

http://java.sun.com

http://xmlns.jcp.org

View Actions

- Actions bei GET-Anfragen ausführen

```
<f:metadata>
  <f:viewParam name="id" value="#{bean.id}"/>
  <f:viewAction action="#{bean.loadPerson}"/>
</f:metadata>
```

- Action-Methode

```
public class Bean {
  private int id;

  public String loadPerson() {
    // Person mit id laden
    return null;
  }
}
```

View Actions: Details

- Standardmäßig in *Invoke Application*

```
<f:viewAction phase="APPLY_REQUEST_VALUES"
    action="#{bean.load}"/>
```

- View-Actions standardmäßig nur bei GET-Anfragen

```
<f:viewAction onPostback="true"
    action="#{bean.load}"/>
```

- Achtung:
  - Navigation immer als Redirect
  - Komponentenbaum noch nicht aufgebaut

Demonstration

View Actions

Beispiel: **https://github.com/jsflive/jsf22-examples/jsf22-view-action**

Dateiupload mit h:inputFile

- Komponente **h:inputFile**

```
<h:inputFile id="file" value="#{bean.file}"/>
```

- Eigenschaft vom Typ **javax.servlet.http.Part**

```
public class Bean {
  private Part file;
  public Part getFile() {return file;}
  public void setFile(Part f) {this.file = f;}
}
```

Demonstration

Dateiupload mit h:inputFile

Beispiel: **https://github.com/jsflive/jsf22-examples/jsf22-input-file**

CollectionDataModel

- **h:dataTable** unterstützt java.util.Collection

```
<h:dataTable value="#{bean.persons}" var="p">
  <h:column>#{p.name}</h:column>
 <h:column>#{p.email}</h:column>
</h:dataTable>
```

```
@Named @RequestScoped
public class bean {
  @Inject
  private PersonRepository personRepository;
  public Collection<Person> getPersons() {
    return personRepository.getPersons();
  }
}
```

Tags für Kompositkomponenten

- Tags für einzelne Kompositkomponenten

```
<facelet-taglib version="2.2">
  <namespace>http://jsflive.at/taglib</namespace>
  <tag>
    <tag-name>collapsible</tag-name>
    <component>
      <resource-id>
        jsflive/collapsiblePanel.xhtml
      </resource-id>
    </component>
  </tag>
</facelet-taglib>
```

Demonstration

Tags für Kompositkomponenten

- Tags für Komponenten aus mehreren Bibliotheken

Beispiel: **https://github.com/jsflive/jsf22-examples/jsf22-composite-component-tags**

Konfigurierbares Ressourcenverzeichnis

- Kontextparameter in **web.xml**

```
<param-name>
  javax.faces.WEBAPP_RESOURCES_DIRECTORY
</param-name>
<param-value>/WEB-INF/resources</param-value>
```

- ... auch für Contracts

```
<param-name>
  javax.faces.WEBAPP_CONTRACTS_DIRECTORY
</param-name>
<param-value>/WEB-INF/contracts</param-value>
```

Verzögerte Ajax-Anfragen

- Ajax-Anfrage wird verzögert
- Nur aktuellste Anfragen wird verarbeitet

```
<h:inputText value="#{bean.product}">
  <f:ajax event="keyup" render="list" delay="300"/>
</h:inputText>
<h:panelGroup id="list">
  <ui:repeat value="#{bean.products}" var="p">
    #{p}<br/>
  </ui:repeat>
</h:panelGroup>
```

Eingabefelder zurücksetzen

- Eingabefelder bei Ajax-Anfrage zurücksetzen

```
<h:form id="form">
  <h:messages id="msgs"/>
  <h:inputText id="v1" value="#{bean.value1}"/>
  <h:commandLink value="+1" action="#{bean.incV1}">
    <f:ajax render="v1" resetValues="true"/>
  </h:commandLink>
  <h:inputText id="v2" value="#{bean.value2}">
    <f:validateLongRange minimum="10"/>
  </h:inputText>
  <h:commandButton value="Save">
    <f:ajax execute="v1 v2" render="msgs v1 v2"/>
  </h:commandButton>
</h:form>
```

Demonstration

Eingabefelder zurücksetzen

- Beispiel

Beispiel: **https://github.com/jsflive/jsf22-examples/jsf22-reset-values**

CDI-View-Scope

- Annotation für View-Scope in CDI

```
@javax.inject.Named
@javax.faces.view.ViewScoped
public class CustomerBean {
  @Inject
  private CustomerService customerService;
  ...
}
```

- Alternative:

Apache MyFaces CODI
@ViewAccessScoped

Status Implementierungen

Oracle Mojarra
2.2.2

Apache MyFaces
2.2-SNAPSHOT

Vielen Dank!

Michael Kurz

irian | THE JAVA EXPERTS

Neugierig?

- Kurz, Marinschek, Müllan:
  JavaServer Faces 2.0, dpunkt.Verlag
- Irian JSF@Work Online-Tutorial
  http://jsfatwork.irian.at
- JSFlive Weblog
  http://jsflive.wordpress.com