

Neuerungen in JSF 2.0

David Eradi

Iseento GmbH
IT-Beratung & Services

Venusweg 1a
90763 Fürth

- Facelets als View-Technologie
- Annotation statt XML
- Composite Components
- Navigation ohne faces-config.xml
- Conditional & Preemptive Navigation
- GET-Requests
- Neue Scopes
- Project Stages & faces-config Ordering
- Bean Validation
- Empty Field Validation / Neue Validatoren
- Error Handling
- Resource Loading
- Client Behaviour
- Native AJAX Unterstützung
- Teilweises State Saving
- System Events
- Quellen

- JSF 2.0 hat eine View Declaration Language - API
- Facelets sind in JSF 2.0 die bevorzugte View-Technologie (und die einzige derzeit)
- JSPs sind deprecated werden aber noch unterstützt

■ Annotationen ersetzen XML

```
<managed-bean>  
  <managed-bean-name>test</managed-bean-name>  
  <managed-bean-class>isento.Test</managed-bean-class>  
  <managed-bean-scope>session</managed-bean-scope>  
</managed-bean>
```

```
@ManagedBean  
@SessionScoped  
public class Test {  
  ...
```

```
@FacesConverter  
@FacesValidator  
@FacesComponent  
@FacesRenderer  
@ResourceDependency  
@ResourceDependencies  
@FacesBehavior  
@ListenerFor  
@ListenersFor  
...
```

- Composite Components sind in JSF 2.0 deutlich einfacher als in JSF 1.2
- Das Hinzufügen von Listener, Validatoren, Converter, ... zur Composite Component ist möglich

```
<!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:composite="http://java.sun.com/jsf/composite">

  <head>
    <title>My First Composite Component</title>
  </head>

  <body>

    <composite:interface>
      <composite:attribute name="who"/>
    </composite:interface>

    <composite:implementation>
      <h:outputText value="Hello, #{cc.attrs.who}!"/>
    </composite:implementation>

  </body>
</html>
```

```
<html xmlns=http://www.w3.org/1999/xhtml
  xmlns:h=http://java.sun.com/jsf/html
  xmlns:greet=http://java.sun.com/jsf/composite/greet>

  ...

  <greet:hello who="World"/>

  ...

</html>
```

- Implicit Navigation (ohne faces-config.xml)

Wenn keine Regel gefunden wird schaut der Navigation Handler ob eine Seite existiert, die der Action entspricht und leitet auf diese weiter.

```
<h:commandLink value="weiter" action="seite2">  
<h:commandLink value=„Anmeldung“ action=„user/registration“>  
<h:commandLink value="weiter" action="seite2?faces-redirect=true">
```

- Conditional Navigation (ohne Beans)

In der faces-config.xml sind if-Abfragen erlaubt:

```
<navigation-case>
  <from-outcome>success</from-outcome>
  <to-view-id>/administration.xhtml</to-view-id>

  <if>#{user.admin}</if>
</navigation-case>
```

Die Regel wird nur aktiv wenn die Bedingung erfüllt ist.

- PreEmptive Navigation

Mit der Methode **getNavigationCase()** kann jederzeit abgefragt werden wohin eine Navigation führen würde.

- GET-Requests können jetzt komfortabel in den JSF-Seiten ausgewertet werden

```
<f:metadata>  
  <f:viewParam name=„user“ value=#{userViewController.userId}/>  
</f:metadata>
```

- Listener können jetzt gesetzt werden

<http://example.isento.net/user/viewUser.jsf?user=5>

```
<f:metadata>  
  <f:viewParam name=„user“ value=#{userViewController.userId}/>  
  <f:event type=„preRenderView“ listener=#{userViewController.loadUser}/>  
</f:metadata>
```

- Parameter können weitergereicht oder gesetzt werden

```
<h:link outcome=„success“ includeViewParams=„true“>  
  
<h:link outcome=„viewUser“>  
  <f:param name=„user“ value=${user.id}/>  
</h:link>
```

- View-Scope
Für die ganze Dauer des Views - Länger als Request, kürzer als Session.
- Flash-Scope
Nur für den Übergang zwischen zwei Views - überlebt Redirects.
- Custom-Scopes
In der faces-config.xml können eigene Scopes definiert werden.

- Project-Stages:

Development, Production, SystemTest, UnitTest

Dank der Project Stages wird klar festgelegt in welchem Abschnitt sich das Projekt befindet. Diese Information kann dann zur Laufzeit abgefragt werden um z.B. die Ausgabe an die jeweilige Phase anzupassen.

- faces-config Ordering

Es ist nun möglich den Dateien in der faces-config Namen zu geben und die Reihenfolge in der sie geladen werden dann anhand dieser Namen zu ordnen.

http://blogs.sun.com/rlubke/entry/jsf_configuration_resource_ordering

- Bean Validation

- Per Annotationen

Automatische Bean-Validation über Standard-Annotationen oder eigene Constraints.

- Im Facelet

Mit [<f:validateBean>](#) kann auch auf der JSF-Seite die Validation umfangreich konfiguriert werden.

```
<h:inputText value="#{bean.foo}">  
  <f:validateBean validationGroups="com.foo.validation.groups.Billable"/>  
</h:inputText>
```

- Empty Field Validation

Felder die null/empty sind werden jetzt standardmäßig auch validiert. Dies kann per Context-Parameter deaktiviert werden.

- Neue Validatoren

- `<f:validateRequired>`

Markiert Pflichtfelder, entspricht `required="true"`.

- `<f:validateRegexp>`

Validation über Reguläre Expressionen, z.B. für eine E-Mail-Adresse:

```
<f:validateRegexp pattern=".+@.+\.[a-z]+"/>
```

- Informativere Fehlerseiten (Zeilennummer der JSF-Seite)
- Neue ExceptionHandler API

Der ExceptionHandler bekommt alle nicht abgefangenen (unchecked) Exceptions, die während des Faces-LifeCycles auftreten.

- ResourceHandler API für
 - JavaScript
 - CSS
 - Images
 - ...
- Ressourcen können in JARs gepackt werden
- ResourceRelocation

Ressourcen können automatisch an bestimmte Stellen in der JSF-Seite gepackt werden. Dafür gibt es die neuen Tags:

```
<h:head>  
<h:body>  
  
<h:outputScript>  
<h:outputStylesheet>
```

- Skripte können in Bundles zusammengefasst werden
- Die Bundles können dann deklarativ zu Komponenten hinzugefügt werden

```
<h:commandLink>  
  <is:confirm message="Sind Sie sicher?"/>  
</h:commandLink>
```

```
<h:commandLink>  
  <f:ajax/>  
</h:commandLink>
```

- JavaScript API `jsf.ajax.request()`

```
<commandButton id="button1" value="submit"  
  onclick="jsf.ajax.request(this,event,  
    {execute:'button1',render:'status',onevent: handleEvent,onerror: handleError});return false;"/>  
</commandButton/>
```

- JSF Tag `<f:ajax>`

```
<h:commandButton">  
  <f:ajax render=„userForm"/>  
</h:commandButton>
```

- Beispiel: <http://java.sys-con.com/node/1352885>

- Mit der Methode **markInitialState()** gibt eine Komponente an, wann ihr Anfangszustand erreicht ist (wird automatisch vom Facelets Tag Handler nach dem Rendern ausgeführt).
- Es wird nicht mehr der ganze State des Komponentenbaums in der Session gespeichert sondern nur die Unterschiede (sogenannter **Delta State**)
- Wenn nötig wird der View einfach neu gerendert und dann der **Delta State** darauf angewendet

Die Session wird dadurch enorm verkleinert

■ Global System Events

- Registrieren mit: `Application.subscribeToEvent();`

```
Application.subscribeToEvent(java.lang.Class, java.lang.Class, javax.faces.event.SystemEventListener)
```

- Auslösen mit: `Application.publishEvent();`

```
Application.publishEvent(javax.faces.context.FacesContext, java.lang.Class, java.lang.Object)
```

■ Component System Events

- Registrieren mit: `UIComponent.subscribeToEvent();`

```
UIComponent.subscribeToEvent(java.lang.Class<? extends SystemEvent> eventClass,  
ComponentSystemEventListener componentListener)
```

oder:

```
<h:inputText>  
  <f:event type="preValidate" listener="#{bean.doSomePreValidation}"/>  
</h:inputText>
```

- Auslösen mit: `Application.publishEvent();`

```
Application.publishEvent(javax.faces.context.FacesContext, java.lang.Class, java.lang.Object)
```

- JAVA Magazin 01/2010 – Seite 30 ff.
- jax TV 2009, „JSF 2.0 A Complete Tour“, JM 01/2010 Heft CD
- <http://andyschwartz.wordpress.com/2009/07/31/whats-new-in-jsf-2/>
- <http://www.javabeat.net/tips/116-new-features-in-jsf-20.html>
- <http://www.ibm.com/developerworks/java/library/j-jsf2fu1/index.html>
- <http://www.ibm.com/developerworks/java/library/j-jsf2fu2/index.html>
- <http://www.ibm.com/developerworks/java/library/j-jsf2fu3/index.html>