anderScore
trust in competence

Was gibt‘s Neues in JSF 2.2?

25.01.2013, A. Arnold

Inhalt

HTML 5 Support

anderScore trust in competence

- Passthrough-Elements
  - HTML – Tags direkt raus schreiben in den Response

HTML5 Support („Big Ticket“)

anderScore trust in competence

- Pass-through attributes

  - Tag-Attribute auf HTML-Tags durchschreiben
  - Generischer Ansatz f. JSF-Tags
  - … und damit Unterstützung f. HTML5 Tag Attribute

  z.B. so:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://java.sun.com/jsf/passthrough"
>
    <h:form>
        <h:inputText value="#{bean.value}" p:placeholder="Enter text"/>
    </h:form>

</html>
```

HTML 5 Support

Oder so mit Taghandler:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
>
    <h:form>
        <h:inputText value="#{bean.value}" >
            <f:passThroughAttribute name="placeholder" value="Enter text" />
        </h:outputText>
    </h:form>

</html>
```

Faces Flows („Big Ticket“)

anderScore trust in competence

- fest definierte Page Flows
  - Wizards etc.
- angelehnt an
  - Spring Webflow
  - MyFaces CODI

Faces Flows („Big Ticket“) cont.

anderScore trust in competence

```
<f:metadata>
    <j:faces-flow-definition>
      <j:initializer>#{someBean.init}</j:initializer>
      <j:start-node>startNode</j:start-node>

      <j:switch id="startNode">
        <j:navigation-case>
          <j:if>#{someBean.someCondition}</j:if>
          <j:from-outcome>fooView</j:from-outcome>
        </j:navigation-case>
      </j:switch>

       <j:view id="barFlow">
        <j:vdl-document>barFlow.xhtml</j:vdl-document>
      </j:view>   ….
     <j:finalizer>#{someBean.finish}</j:finalizer> ….
```

AJAX Fileupload („Big Ticket“)

anderScore trust in competence

- Basiert auf Servlet 3.0 multipart-Support

- Keine direkte Unterstützung in HTML
  -> iframe als Workaround für AJAX-Download

```
<h:form prependId="false" enctype="multipart/form-data">

    <!-- The new regular file upload component -->
    <h:inputFile id="fileUpload" value="#{someBean.file}" />

    <h:commandButton value="Upload" />
</h:form>
```

CSRF – Protection („Big Ticket“)

anderScore trust in competence

- Cross-Site Request Forgery

- Szenario:
  - Angriff innerhalb einer Webanwendung mit authorisiertem User
  - Ziel: Datenänderung durch URL-Manipulation

  - Angreifer schiebt eine bösartige URL unter…
    - z.B. in einem Forum-Post mit <img> URL

    <img
    src="http://bank.example.com/withdraw?account=Alice&amount=1000000&for=Eve">

  - … auf die das Opfer klickt
    - Hat bei http://bank.example.com einen Account
    - … Opfer ist noch eingeloggt …

CSRF – Protection („Big Ticket“) … continued

- Schutz gegen Cross-Site Request Forgery- Angriffe
  - Bisher eingschränkte Absicherung durch
    - serverseitigen State +
    - ViewState ID als hidden Parameter in postback-Requests

- Lösung in JSF 2.2
  - Wert der ViewState ID wird zufällig gewählt
  - Client State encryption angeschaltet per Default
  - Zusätzliches Token in non-postback Requests

GET Support (1)

anderScore trust in competence

Bereits seit JSF 2.0, aber

- Nur Handling von Request-Parametern

```
<!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="…"  xmlns:f="…">
  <f:metadata>
    <f:viewParam id="id" name="item" value="#{catalog.item}"/>
  </f:metadata>
  <h:body>
    You requested catalog item: <h:outputText value="#{catalog.item}"/>
  </h:body>
</html>
```

- Keine GET-basierten Actions
  - Workaround: Listener auf „preRenderView“ Event

GET Support (2)

anderScore trust in competence

- Neue Komponente: **viewAction** (aus Seam 3)

```
<f:metadata>
        <f:viewAction action="#{someBean.someAction}" />
</f:metadata>
```

- Vorteile:
  - viewActions früh getriggert, vor Aufbau des Komponentenbaums
  - Überall wo GET denkbar ist
  - Implizite und explizite Navigation („navigation-rule“s)
  - non-Faces (initiale) Requests und Faces Request (postback)

Injection (fast) überall

anderScore trust in competence

- Dependency Injection auf allen JSF-Artefakten
  - bisher nur Managed Beans als Injection Target
  - Jetzt auch:
    - Converter
    - Validator
    - UIComponent
    - Behavior
    - Andere per Deklaration in face-config.xml

CDI – @ViewScoped

- Bisher nur JSF-Variante von ViewScoped
  - Dann aber kein CDI
  - Oder spez. CDI Extensions in Seam o. Apache CODI

- Jetzt CDI-kompatibel
  - per CDI Extension
  - javax.faces.bean.ViewScoped wird deprecated (angekündigt in javadocs)
  - GET Request in selbe View liegen auch im ViewScope

```
import javax.inject.Named;
import javax.faces.flow.ViewScoped;

@Named
@ViewScoped
public class Foo {
    // ...
}
```

Facelet Component Tag per Annotation erzeugen (1)

anderScore trust in competence

```
@FacesComponent(value="components.CustomComponent", createTag=true)
public class CustomComponent extends UIComponentBase {

    @Override
    public String getFamily() {
        return "my.custom.component";
    }

    @Override
    public void encodeBegin(FacesContext context) throws IOException {
         String value = (String) getAttributes().get("value");
         if (value != null) {
            ResponseWriter writer = context.getResponseWriter();
            writer.write(value.toUpperCase());
        }
    }
}
```

Facelet Component Tag per Annotation erzeugen (2)

anderScore trust in competence

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:test="http://java.sun.com/jsf/component"
>
    <h:body>
        <test:customComponent value="test"/>
    </h:body>
</html>
```

Facelet Component Tag per Annotation (3)

anderScore trust in competence

- Konfiguration per Annotationsattribute:
  - createTag – wenn „true“, wird ein neues Tag erzeugt
  - tagName – Name des Tags, sonst Klassenname der Komponente, erstes Zeichen lowercased
  - namespace – Namespace des Tags, sonst http://java.sun.com/jsf/component

Programmatische Konfiguration (1)

- bisher z.B. per ViewHandler, aber problematisch (schlägt fehl, wenn JSF noch nicht initialisiert)

- Neuer Callback, daher keine timing Probleme
  - Callback-Method mit **@FacesConfigDocumentPopulator** annotieren
  - Nimmt ein DOM Document entgegen

Programmatische Konfiguration (2)

anderScore trust in competence

Callback

```
@FacesConfigDocumentPopulator
public void doConfig(Document document) {

    String ns = document.getDocumentElement().getNamespaceURI();

    Element applicationEl = document.createElementNS(ns, "application");
    Element viewHandlerEl = document.createElementNS(ns, "view-handler");

    viewHandlerEl.appendChild(
        document.createTextNode(MyViewHandler.class.name())
    );

    applicationEl.appendChild(viewHandlerEl);
    document.appendChild(applicationEl);
}
```

API – Zugang zu Factories / neue Wrapper

anderScore trust in competence

- JSF holt Factories über den „FactoryFinder“
  - FactoryFinder.getFactory (FactoryFinder.FACELET_FACTORY)

- Neu über FactoryFinder in Standard-API:

  - FaceletFactory → FactoryFinder.FACELET_FACTORY
  - FlashFactory → FactoryFinder.FLASH_FACTORY)

- Neue Wrapper:
  - Verhalten anpassen, durch überschreiben einzelner Methoden
  - FlashWrapper f. Flash
  - ViewHandlerWrapper f. ViewHandler
  - ExceptionHandlerWrapper f. ExceptionHandler
  - usw.

CompositeComponents

anderScore trust in competence

- Können jetzt per neuer API dynamisch erzeugt werden

Quellen

- http://jdevelopment.nl/jsf-22/#802
- http://www.oracle.com/technetwork/articles/java/jsf22-1377252.html
- Wikipedia