JSF, Conversion, Validation AJAX and Navigation

JSF Slides #3

Attached objects

Components in server side UI tree may have attached objects, i.e. non-UIComponents attached to UIComponents

- **Converters**
- **Validators**

Conversion

Submitted values converted and stored in components (in tree)
- Mostly automagically

If more specific needs, JSF Standard converters for
- DateTime, Number, Boolean, Byte, Character, Double, Float, Integer, Long, Short, BigDecimal, BigInteger

```
<!-- Using h- anf f- tags from JSF -->
<h:inputText id="date" value="#{testBean.date}" required="true">
    <f:convertDateTime type="date" dateStyle="long"/>
</h:inputText>
```

If very specific needs create custom converter

Custom Conversion

Converting phone numbers

```
@FacesConverter(value = "pnumbConverter")
public class PhoneNumberConverter implements Converter {
  public Object getAsObject(FacesContext context, UIComponent
                            component, String value) {...}

  public String getAsString(FacesContext context, UIComponent
                           component, Object value) {...}
}
```

Usage

```
<h:inputText value="..." ....>
   <f:converter converterId="pNumbConverter"/>
</h:inputText>
```

Validation

Principle: All layers should validate input data, validate client-side and server side
- Client side : HTML5/CSS3/JavaScript client side validation
- Server side: Two different ways to validate

JSF Validation, part of JSF specification

Beans Validation a different specification
- JSF validation targeting just JSF, Bean Validation targets "any" kind of Java application (also Java SE), prefer ...

JSF Validation

Tags...

```
<f:validateLength maximum="8" minimum="1"  />
<f:validateRegex ..... />
    ...

<!-- In a page -->
<h:inputText id="txtData" value="#{bean.data}" required=
                    "true" validatorMessage = "Must be 1-8 chars" >
   <f:validateLength minimum="1" maximum="8"   />
</h:inputText>
<h:message for="txtData" showSummary="true" showDetail="false"
                                        style="color: red" />
```

..and many more options

JSF Custom Validation

Validate email

```
@FacesValidator("edu.hajo...EmailValidator")
public class EmailValidator implements Validator{
    //...
}
```

Usage

```
<h:inputText id="email" value="#{user.email}"
           size="20" required="true" label="Email Address">
    <f:validator validatorId="edu.hajo...EmailValidator" />
</h:inputText>
```

Bean Validation

Works seamless with JSF, use annotations (validation errors will end up in <h:message(s)../>)

```
@Named
@SessionScoped
public class ValidateBean implements Serializable {

    //@Digits(integer=2, fraction=0)
    @Min(value = 1, message = "To small")
    @Max(value = 10, message = "To big")
    // See also @Min.List
    private int value;
    @NotNull
    @Size(min = 1, max = 8, message = "Must use 1-8 chars")
    private String data;
    // Also @Past @Future
    @Pattern(regexp = "[A_Z][a-z]*")
    private String pattern;
```

JSF and Bean validation not in same phase, JSF first then Beans (if not JSF failed, then skipped)

Validation Messages

Possible to store localized messages in ValidationMessages.properties file (exact that name, possible suffix for language _en, _sv!)

```
// ValidationMessages.properties
user.password =  Password name must be 3-20 characters
…
```

Location of file in NetBeans:
Other Sources > src/main/resources/default package
Location in war: WEB-INF/classes

Usage in code

```
@Size(min = 3, max = 20, message = "{user.password}")
private String password;
```

Bean Validation Advanced

Bean validation is extremely customizable! Possible to define advanced validations

- Creating own annotations for attributes, example @OrderNumber
- Creating validation annotations for complete class (object)
- Validation groups
- Different stages, i.e. validation rules vary over time

Client Side Behaviours

JSF has a generic solution for associating client-side scripts (JavaScript) with UI components

- AJAX, Client-side validation, DOM and style manipulation, animations and visual effects, Alerts and confirmation dialogs, Tooltips and hover content, Keyboard handling, Deferred (lazy) data fetching, Integration with 3rd party client libraries, Client-side logging

Using Ajax with JavaServer Faces Technology

JSF and AJAX

JSF is default "full page load"
- Suitable for many application
- Real need for AJAX ...?? If so ...

Three ways to add AJAX behaviour
- **Programmatic** AJAX using raw Javascript. The standard Javascript resource library bundled with JSF (jsf.js). Global jsf-object, ... we avoid!
- Using <f:ajax> tag, **declarative** AJAX, we use some...!
- Use a higher level component suite (PrimeFaces, OpenFaces, ...), upcoming, no explicit AJAX coding, prefer!

JSF AJAX tag

<f:ajax> will add AJAX behavior to component(s)
- Nested inside component tag or..
- Enclose component tag(s)
- Both possible (behavior: Union of all <f:ajax>)

Many attributes defining
- Which component (id) AJAX applies to (execute)
- What event (onclick, ...) (event)
- Where to render (insert) result
- Others: listener, immediate, disabled

JSF AJAX tag cont.

Some special values to use for “execute” and “render” attributes of <f:ajax> elements (there are more)

execute= “@...”
- @all, every component is submitted/processed
- @form, submit/process entire form

render = “@...”
- @all, all components rendered
- @form, render entire form

```
<h:form>
        <h:inputText id="myinput" value="Text: #{userBean.name}"/>
        <h:outputText id="outtext" value="Echo: #{userBean.name}"/>
        <h:commandButton id="submit" value="submit">
                <f:ajax execute=”@form” render=”outtext”/>
        </h:commandButton>
</h:form>
```

JSF Navigation Model

Two possible techniques (samples)

Implicit navigation

- In pages or Java code
- Useful for links

Rule based navigation (user defined)

- In **faces-config.xml**
- Variations
  - Static
  - Dynamic
  - Conditional

Late in request cycle (second last)

Implicit Navigation

Strings interpreted as filename (no xhtml suffix). Using action

```
<!-- Navigate to page confirm.xhtml -->
<h:commandButton … action=”confirm” ../>

<!-- Navigate to outcome of method -->
<h:commandButton … action=”#{bean.navigate}” ../>
```

In bean

```
public String navigate(){
      ..
   return "aFileNameWithoutSuffix"; // If null no navigation
}
```

Or just a link using outcome

```
<h:link value="Edit" outcome="personDetail">
```

Rule Based Navigation

Rules applied in written order, first success wins

```
<!-- faces-config.xml -->
 <navigation-rule>
     <from-view-id>/viewProducts.xhtml</from-view-id>
     <navigation-case>
         <from-outcome>onViewCart</from-outcome>
         <to-view-id>/viewCart.xhtml</to-view-id>
     </navigation-case>
     <!-- More cases -->
</navigation-rule>
```

Variations
- Static, hard coded
- Dynamic, from bean method result
- Conditional, possible to avoid "first success wins"

Redirects

Possible to do redirect using a predefined string

- Parameters don't survive

```
<!-- Also possible as return value from bean -->
<h:commanLink ... action="to?faces-redirect=true".../>
```

Or in faces-config.xml

```
<navigation-rule>
   <from-view-id>/searchForm.jsp</from-view-id>
   <navigation-case>
      <from-outcome>submit</from-outcome>
      <to-view-id>/searchResults.xhtml</to-view-id>
      <redirect/>
   </navigation-case>
</navigation-rule>
```

View Parameters and Redirect

Parameters don't survive redirects.
If using view parameters possible to let JSF automatically add "what target page wishes"
- Target side as before using <f:viewParam>

Sources side using includeViewParams at action
- All data in target side (like ${client.id}) appended to query string
- Navigation subsystem will try to match (using source and dest)

```
// In Bean
return action="target?faces-redirect=true&amp;includeViewParams=true"

// In page
<h:commandButton ... action="target?faces-redirect=true&amp;
includeViewParams=true" … />
```

JSF Navigation Errors

Navigation system will try to resolve all navigational cases

- If not possibly will get error message (in page)