

AWT

4) JSF Lifecycle, Event handling, data binding, i18n

Emmanuel Benoit
Fall Term 2016-17

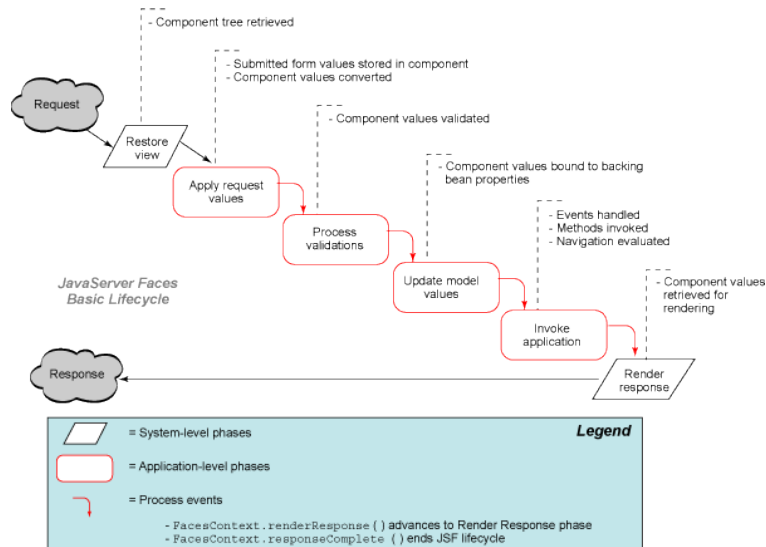
Table of Contents

- Life-cycle
 - Basic JSF Life-cycle
 - Conversion and validation
 - Invoke Application
 - Actors in the JSF process
- Internationalization - I18n
 - Motivations
 - I18n in Java (very short presentation)
 - Change Language
 - I18n in JSF
- Conclusion

Life-cycle

Basic JSF Life-cycle

Basic JSF Lifecycle



Life-cycle

- ▶ **Create the component tree**
- ▶ **Map values to the component nodes (corresponding to request parameters)**
 - ▶ Request parameters are stored in the component themselves
- ▶ **Process validations** Validate each of the obtained values
- ▶ **Update Model Values** Value is copied inside the corresponding backing beans.
- ▶ **Invoke Application** : Event handling and actions are executed.
- ▶ **Render Response**
 - ▶ A new component tree is created (if needed by navigation)
 - ▶ Backing beans values are transferred back to Components
 - ▶ Components are rendered

Conversion and validation

- ▶ **Submitted form values are stored inside the component**
- ▶ **Value is converted**
 - ▶ The component contains one type of input
 - ▶ Input is always sent as a "String"
 - ▶ JSF must convert the value into the right type
 - ▶ Conversion may be implicit or explicit

Apply Request Value (Cont.)

Use Converters

► Can be the default JSF converters

```
<!-- UserRegistration.xhtml -->
<h:inputText id="age" value="#{UserRegistration.user.
→age}">
    <f:converter id="javax.faces.Short"/>
</h:inputText>
```

► Or a Date default converter

```
<!-- UserRegistration.xhtml -->
<h:inputText id="birthDate"
    value="#{UserRegistration.user.birthDate}">
    <f:convertDateTime pattern="MM/yyyy"/>
</h:inputText>
```

Define your own converter

► Create a Converter class

```
import javax.faces.convert.Converter;
public class PhoneConverter implements Converter {
    public Object getAsObject(FacesContext context,
        UIComponent component, String value) { ... }
    public String getAsString(FacesContext context,
        UIComponent component, Object value) { ... }
}
```

► Declare its use in the faces-config.xml file

► Use the declared name in the XHTML File

```
<h:inputText id="phone"
    value="#{UserRegistration.user.phone}">
    <f:converter converterId="arcmind.PhoneConverter" />
</h:inputText>
```

Validation

► Once the value is inside each component, it is validated

```
<h:inputText id="age" value="#{UserRegistration.user.
→age}">
    <f:validateLongRange maximum="150"
        minimum="0"/>
</h:inputText>
```

► If conversion or validation fails:

- A message is generated (printed in the message tag)
- The life-cycle goes directly to the rendering step
- The page is rendered directly (other steps are not executed)
- Values are never transferred to the backing beans in this case.

► If conversion and validation succeed:

- Value is transferred to the corresponding Backing bean.
- Backing beans "getter" methods are executed using values stored in the components



Invoke Application

Invoke Application: Event handling and actions

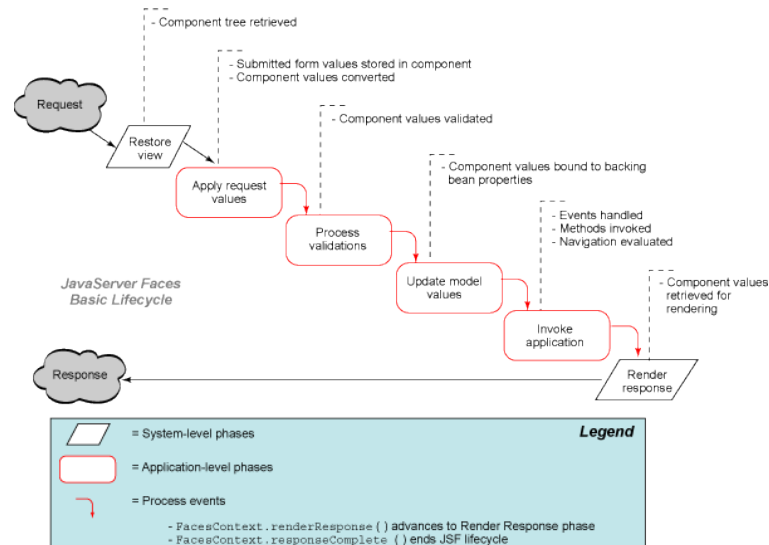
- ▶ **After the value is transferred to ALL backing beans**
- ▶ **Events are fired by the component**
 - ▶ Value Changed Events : for input fields
 - ▶ Action Event : for command components
- ▶ **EL expressions corresponding to actions are computed**
 - ▶ They can use all the information stored inside the backing beans
- ▶ **Action values are used to fire a navigation rule**
 - ▶ The View page is computed according to the returned value

Rendering

- ▶ **The values in the beans are stored back in the tree**
 - ▶ The component accesses the getter method of the bean
- ▶ **Each component know how it should be rendered**
- ▶ **Some render themselves directly**
 - ▶ Using the encoding functions: `encodeBegin(...)`, `encodeChildren(...)` and `encodeEnd(...)`
- ▶ **Some use an external renderer (or many depending on the application)**
 - ▶ The encoding and decoding functionalities are delegated to a "Renderer" Class.

```
public class FieldRenderer extends Renderer {
```

Basic JSF Life-cycle (the same picture)



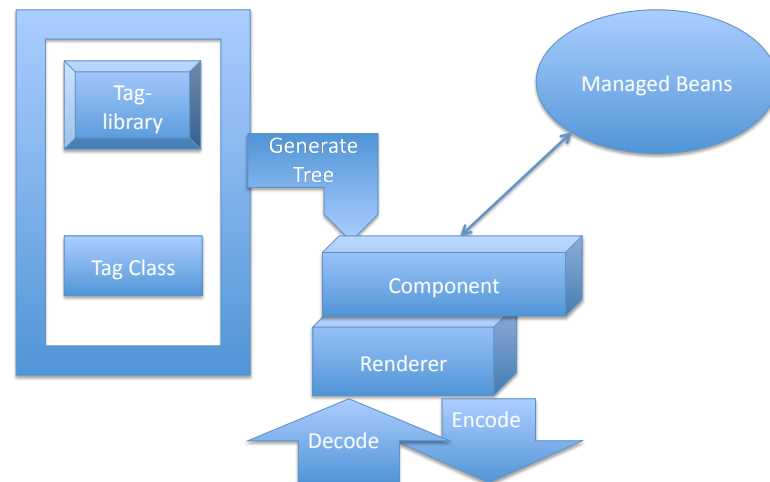
Actors in the JSF process

- ▶ **Components :**
 - ▶ Compose the component tree (nodes have one parent and can have many children)
 - ▶ Are the center piece of our puzzle
- ▶ **Managed Beans**
 - ▶ Contain the values stored by the application
 - ▶ Contain the link with the business layer
- ▶ **Renderer**
 - ▶ “decode” the form input
 - ▶ “encode” the component

Actors in the JSF process (Cont.)

- ▶ **Tag-lib**
 - ▶ xml definition of the tag and its arguments
- ▶ **Tag class**
 - ▶ The class instanciated by the tag lib.

Actors



Internationalization - I18n

Motivations

i18n in Java (very short present)

Internationalization ?

- ▶ **Multilingual web applications**
 - ▶ Work of programmers should be used anywhere in the world
 - ▶ Translation should not require any informatics knowledge
- ▶ **Structure**
 - ▶ Web application without any text,
 - ▶ Data Base designed to handle multilingual texts,
 - ▶ Static texts are stored in resource bundles.
- ▶ **Language**
 - ▶ Automatically recognized from the browser,
 - ▶ Comparison between the site and the browser,
 - ▶ The user can also change the desired language.
- ▶ **Priority**
 - ▶ browser identification (lowest)
 - ▶ Locale in the session
 - ▶ Change using an event (highest)

i18n in Java

- ▶ **A Locale object contains i18n configurations**
 - `public Locale(String language)`
 - `public Locale(String lang, String country)`
 - `public Locale(String lang, String ctry, String variant)`
- ▶ **Resource bundles:**
 - ▶ Provide facilities for storage and retrieval of all locale-specific information, independently from the application logic
 - ▶ Allow to support multiple locales in a single application
 - ▶ Allow to extend internationalization easily
- ▶ **The Java Resource bundles classes are:**
 - ▶ `ResourceBundle` contains locale-specific objects.
 - ▶ `ListResourceBundle` abstract subclass of `ResourceBundle`
 - ▶ `PropertyResourceBundle` is a concrete subclass of `ResourceBundle` (property files).

Resource Bundle (Example)

```
public class MyResourceBundle extends ResourceBundle {
    private String keys = "Msg1_Msg2_Msg3";
    public Object handleGetObject(String key) {
        if (key.equals("Msg1") return "Hello_world!";
        if (key.equals("Msg2") return "Hello_i18n!";
        ...;
        return null;
    }
    public Enumeration getKeys() {
        return new StringTokenizer(keys);
    }
}
```

Resource Bundle

► Retrieve localized information

To retrieve a localized value for a given key, you should use one of the methods

- getObject,
- getString or
- getStringArray

from the class ResourceBundle:

- public Object getObject(String key)
 - first tries to obtain the value using handleGetObject.
 - If not successful, it calls the getObject method of the parent resource bundle, assuming it is not null.
 - The other two methods are convenience methods that cast the object returned.

Change Language

Property Resource Bundles

► A Property Resource Bundle is a collection of text elements stored in a property file.

- A property file is a text file containing properties. Therein:
- A property is specified as "key = value" or "key : value"
- Line beginning with "!" or "#" are comments
- "\\" is used to indicate line continuation

```
# File name: I_Classes.properties
ApplicationTitle=Classes in dept. I
DisplayButtonText=Display
EndButtonText=Exit
I1=I1a, I1b, I1c, \
    I1p, I1q, I1r
I2=I2a, I2b, I2c, \
    I2p, I2q, I2r
I3=I3SE, I3TM, I3WI, I3p, I3q
I4=I4t, I4v, I4w
```

I18n in JSF

I18n in JSF

- ▶ **Declare the supported Locales**
 - ▶ In the `faces-config.xml` file
- ▶ **Create for each supported language a Property file**
 - ▶ `project/src/ch/bfh/toto/Toto.properties`
- ▶ **Replace any output string in the XHTML files with a message**

```
<h:outputText value="#{bundle.title}" />
```

or simply

```
#{bundle.title}
```
- ▶ **Give the possibility for the user to change language**

Language selection in HTTP

- ▶ **Browser sends its preference in the HTTP Header**
 - ▶ Which languages are supported
 - ▶ Which formats are supported
 - ▶ ...

```
GET http://cms.hta-bi.bfh.ch/typo3/index.php HTTP/1.1
```

```
Host: cms.hta-bi.bfh.ch
```

```
...
```

```
Accept-Language: fr, fr-ch;q=0.83, en;q=0.66, en-us;q=0.50, ↘  
→de;q=0.33, de-ch;q=0.16
```

```
Accept-Encoding: gzip, deflate, compress;q=0.9
```

```
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
```

```
Keep-Alive: 300
```

```
Proxy-Connection: keep-alive
```

Declare locales in `faces-config.xml`

```
<faces-config>  
  <application>  
    <locale-config>  
      <default-locale>en</default-locale>  
      <supported-locale>fr</supported-locale>  
      <supported-locale>de</supported-locale>  
      <supported-locale>es</supported-locale>  
    </locale-config>  
  </application>  
...
```

```
...
```


Create a property file

- ▶ **Create files in your src arborescence**

- ▶ Example
 - project/src/ch/bfh/jsf/Resources.properties
 - ▶ project/src/ch/bfh/jsf/Resources_fr.properties

```
#  
# This file is used to store localized expressions  
# Autor: E.Benoist  
#  
title=Hello world  
message=Try to find out what I think  
congratulation=Congratulation
```

Change Expressions in the XHTML files

- ▶ **Load the bundle, according to the Locale**

```
<f:loadBundle basename=" carstore.bundles.Resources"  
              var=" bundle" />
```

- ▶ **Write a message**

```
<h:outputText styleClass=" maintitle"  
              value=" #{bundle.chooseLocale}" />
```

Messages can contain a variable part

- ▶ **Some messages need to include a variable part**

- ▶ *Félicitation, vous avez trouvé que le numéro était le 10.*
- ▶ *Gratulation, Sie haben gefunden, dass die Nummer die 10 war.*

- ▶ **We can write such a sentence in the messages.properties file**

```
felicitation=Gratulation, ... , dass die Nummer die {0} war.
```

Messages can contain a variable part (Cont.)

- ▶ **Reference in the XHTML:**

```
<h:outputFormat value=" #{msg.success_text}" >  
  <f:param value=" #{numberBean.userNumber}" />  
</h:outputFormat>
```

Change the Locale

▶ Reacting to an event

- ▶ Generate an ActionEvent in a form (works also with a button):

```
<d:map id="worldMap" current="Europe" immediate="↵"  
→true" action="storeFront"  
actionListener="#{carstore.chooseLocaleFromMap}" >
```

- ▶ Or

```
<h:commandLink id="NAmerica" action="storeFront"  
actionListener="#{carstore.chooseLocaleFromLink}" >
```

- ▶ In the Bean

```
public void chooseLocaleFromMap(ActionEvent actionEvent) {  
    AreaSelectedEvent event = (AreaSelectedEvent) actionEvent;  
    String current = event.getMapComponent().getCurrent();  
    FacesContext context = FacesContext.getCurrentInstance();  
    context.getViewRoot().setLocale((Locale) locales.get(current));  
    resetMaps();  
}
```

Return a Localized message

▶ Messages can be returned by a bean

- ▶ Managed beans are used for this purpose
- ▶ They should be internationalized

▶ Idea

- ▶ Write all messages in a ResourceBundle (with a property file)
- ▶ Load this class in your program
- ▶ Answer to getXXX with a localized message.

Example of Localized Message

▶ In the constructor of the bean

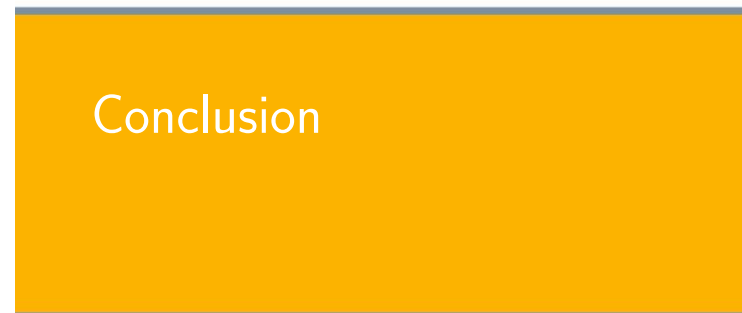
```
FacesContext context = FacesContext.getCurrentInstance↵  
→();  
ResourceBundle data = null;  
Enumeration keys = null;  
components = new HashMap();
```

```
// load the labels
```

```
resources =  
    ResourceBundle.getBundle(CarStore.↵  
→CARSTORE_PREFIX +  
        ".bundles.Resources",  
        context.getViewRoot().getLocale());
```

▶ When a message is requested

```
optionLabel = resources.getString(optionKey);
```



Conclusion

- ▶ **JSF Lifecycle**
 - ▶ Creation of components in the component tree
 - ▶ Population of the tree
 - ▶ THEN transfer to the managed beans
 - ▶ And finally event handling and navigation
 - ▶ Different actors: Components, Managed Beans, Converters and Validators, Taglib and Tag class, ...
- ▶ **i18n**
 - ▶ Straightforward for a JSF application
 - ▶ Support out of the box
 - ▶ Should never include a string inside any programm

References

- ▶ <http://www-128.ibm.com/developerworks/java/library/j-jsf3/>