Berner Fachhochschule-Technik und Informatik

Advanced Web Technologies
5) JSF The View Part

Dr. E. Benoist

Fall Semester 09-10

Table of Contents

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 (higest)

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

- **Retreve localized information**
  To retrieve a localized value for a given key, you should use one 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 casts the object returned.

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
```

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
```

I18n in JSF

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

  <h:outputText value="#{bundle.title}" />
- **Give the possiblity for the user to change language**

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 JSP 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}" />
```

Change the Locale

- **Reacting to an event**
  - Generate an ActionEvent in a form (works also with a button):

```
<d:map id="worldMap" current="NAmericas" 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**
  - Manged beans are used for this purpose
  - They should be internationalized
- **Idea**
  - Write all messages in a MessageBundle (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);
```

Navigation Rules

- **Design the way to surf from one page to the next**
  - Form and link contain `action` attribute

```
<h:commandLink id="SAmerica" action="storeFront">
  <h:outputText value="#{bundle.spanish}" />
</h:commandLink>
```

  - This link should correspond to an entry in the `faces-config.xml`

```
<navigation-rule>
  <from-view-id>/chooseLocale.jsp</from-view-id>
  <navigation-case>
    <from-outcome>storeFront</from-outcome>
    <to-view-id>/storeFront.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Navigation Rules II

- **Value can be returned by a program**
  - In the JSP we have a button

    <h:commandButton action="#{guess.testValue}" value="Test" />
  - in the managed bean guess

```
public String testValue(){
  if(guess==value)
      return FOUND;
  if(guess == -1)
      hint = HINT_NOT_VALID;
  // ...
  return NOT_FOUND;
}
```

Navigation Rules III

- In the `faces-config.xml`

```
<navigation-rule>
  <from-view-id>/guess.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/congratulation.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>failure</from-outcome>
    <to-view-id>/guess.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Navigation Rules IV

- **Writen URL (in the browser) is the called one**
  - Convenient since it answer a request to this page
  - Problematic, since we are not any more in this JSP file
- **Solution**
  - Create a redirect rule

```
<navigation-rule>
  <from-view-id>/guess.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/congratulation.jsp</to-view-id>
    <redirect />
  </navigation-case>
  ...
```

Navigation Rules V

- **Some redirections are common to any file: Solution wildcards**

```
<navigation-rule>
  <from-view-id>/*</from-view-id>
  <navigation-case>
    <from-outcome>error</from-outcome>
    <to-view-id>/error.jsp</to-view-id>
    <redirect />
  </navigation-case>
</navigation-rule>
```

Navigation Rules VI

- **Using** `from-action`
  - Suppose for one page, we have two possible actions: `startOverAction` and `answerAction` that may return the same value `again`

```
<navigation-case>
  <from-action>#{quiz.answerAction}</from-action>
  <from-outcome>again</from-outcome>
  <to-view-id>/again.jsp</to-view-id>
</navigation-case>
<navigation-case>
  <from-action>#{quiz.startOverAction}</from-action>
  <from-outcome>again</from-outcome>
  <to-view-id>/index.jsp</to-view-id>
</navigation-case>
```

Events Handling

- **Two types of events**
  - *Value Changed* (`javax.faces.event.ValueChangeEvent`) observes changes to the user interface component property
    - Expanding a tree,
    - changing the text of a field
  - *Action* (`javax.faces.event.ActionEvent`) Observes the activation of a descendent of a UICommand
    - buttons
    - hyperlinks

Events Handling (Cont.)

- **Example I:** Value Changed listener on an input text field

```
<h:inputText id="userId" value="#{login.userId}">
  <f:valueChangedListener type="ch.bfh.jsf.UserLoginChanged"
<h:/inputText>
```

- **Example II:** Action event listener on a command button

```
<h:commandButton id="login" commandName="login">
   <f:actionListener type="ch.bfh.jsf.LoginActionListener" />
</h:commandButton>
```

Action Event

- **Generated by a button or an hyperlink**
  - `h:commandButton`
  - `h:commandLink`

```
<h:commandLink id="NAmerica" action="storeFront"
               actionListener="#{carstore.chooseLocaleFromLink}">
  <h:outputText value="#{bundle.english}" />
</h:commandLink>
```

- **Handled by a managed bean**

```
public void chooseLocaleFromLink(ActionEvent event) {
  String current = event.getComponent().getId();
  FacesContext context = FacesContext.getCurrentInstance();
  context.getViewRoot().setLocale((Locale) locales.get(current));
  resetMaps();
}
```

ActionEvent with an image

- **In the JSP**

```
<h:commandButton image="toto.jpg"
    actionListener="#{rushmore.listen}"
    action="#{rushmore.act}"
```

- **In the java file**

```
public void listen(ActionEvent e){
  FacesContext context = FacesContext.getCurrentInstance();
  String clientID = e.getComponent().getClientID(context);
  int x = new Integer((String)requestParams.get(clientId+".x")).int
  int y = new Integer((String)requestParams.get(clientId+".y")).int
 ...
  outcome = ...
}
public String act(){ return outcome; }
```

Value Changed Events

- **Called if an element has been changed**
- **in the jsp file**

```
<h:selectOneMenu value="#{form.country}" onchange="submit()"
    valueChangeListener="#{form.countryChanged}">
   <f:selectItems value="#{form.coutryNames}" />
</h:selectOneMenu>
```

- **In the managed bean**

```
public void countryChanged(ValueChangeEvent event){
  FacesContext context = FacesContext.getCurrentInstance();
  if((String)event.getNewValue().equals("US"))
    context.getViewRoot.setLocale(Locale.US);
  else
    context.getViewRoot.setLocale(Locale.FR);
}
```

Value Changed event

- **Has three main methods**
  - `getComponent()` Returns the input component that triggered the event.
  - `getNewValue()` Returns the current local value of the source UIComponent.
  - `getOldValue()` Returns the previous local value of the source UIComponent.

Conclusion

- **Internationalisation support native in JSF**
  - Bundle support of Java
  - Texts automatically generated by a localized bundle
- **Navigation Rules**
  - Allow to navigate between pages
  - Separate from pages
  - Just the controler knows which result leads to which page
  - Separation of layers
- **Event Handling**
  - Can react to small changes
  - More oriented toward “Components”
  - More fine-tuning

References

- Core Java Server Faces
  David Geary and Cay Horstmann
  Sun Microsystems