Internet Application Project: Java Server Side

420-625-DW

Multi-Language Support

Multi-language support

- Goal: Support multiple languages in as easy a way as possible
- We are in Quebec, so this is particularly relevant!

Two sorts

- There are two sorts of multi language issues:
  - Content issues
    - Storing titles in two languages and connecting them to the same content
    - This is a database problem
  - Static issues
    - Fixed words such as “welcome” or “shopping cart”
    - This is a JSF problem

Content issues

- To store content in multiple languages is mostly a *database* issue
- What are some options?

Database solutions

- One database/schema per language
- Extra columns within the same tables (e.g. English_title, French_title)
  - Bad if you have more than 2 languages, but possibly acceptable
- One schema with lots of tables and some joins

Database solutions

Movie

Id
ProductionYear
Director name

LanguageMovie

Title
Description
Language
Movie_Id

Database solutions

Id	Production Year	Director name
1	2000	Jane Doe
2	1995	John Doe

Id	Title	Description**	Language	Movie_id (foreign key)
1	Hola!	Una película sobre saludos.	ES	1
2	Bonjour!	Un film sur les salutations.	FR	1
3	Hello!	A movie about greetings.	EN	1

** Blame google translate if you don't like these translations!

Database solutions

Id	Production Year	Director name
1	2000	Jane Doe
2	1995	John Doe

Id	Title	Description**	Language	Movie_id (foreign key)
1	Hola!	Una película sobre saludos.	ES	1
2	Bonjour!	Un film sur les salutations.	FR	1
3	Hello!	A movie about greetings.	EN	1

In many cases, it would make sense for Language to also be a separate foreign key, instead of being inline here (makes for better data)

Database solutions

Id	Production Year	Director name
1	2000	Jane Doe
2	1995	John Doe

Id	Title	Description**	Language	Movie_id (foreign key)
1	Hola!	Una película sobre saludos.	ES	1
2	Bonjour!	Un film sur les salutations.	FR	1
3	Hello!	A movie about greetings.	EN	1

One thing to be cautious about: This approach will potentially result in many joins!

Multi language : JSF

JSF will make managing multi language or multi-culture support quite easy for static pages

Resource Bundle

A resource package is useful for avoiding hard coding strings

When we create a resource package, we can refer to elements within the resource package as if they were beans

Resource Bundle : How to

1)Create a folder (or find existing folder): src\main\resources

2)Within resources, create a folder structure as you would for a package and then within this, create a file *someName.properties* (for example “messages.properties”

This file should be a pair of values, one per line, representing a dictionary.

Note that this file is *case-sensitive*

```
greeting = HelloWorld!
buy = Buy now
```

Resource Bundle : How to

3)Inside the faces-config.xml file (within WEB-INF), add config information specifying this resource file:

Within an <application> tag,

```
 <resource-bundle>
     <base-name>shouldMatchFolderStructure.messages</base-name>
     <var>nameOfBean</var> <!-- how you'll use it -->
  </resource-bundle>
```

Example folder structure

src\main\resources\foo\bar\file.properties

In this case, the <base-name> tag should have

foo.bar.file

In the xhtml

Now, within the xhtml, you can refer to the dictionary you’ve defined

#{nameOfBean[‘nameOfParameter’]

For example

#nameOfBean[‘greeting’]

Internationalization

The resource bundle is a good programming practice to start with, as all of your string literals are now in one place

To make the code internationalizable, you simply need to add one file per language and then add some code to *set* the language/culture

Start by creating 1 file per language/culture

file_language.properties (e.g. file_fr.properties)

Or

file_culture.properties (e.g. file_fr_FR.properties)

Add to the faces-config.xml

Then add to the faces-config.xml file, within the <application> tag:

```
<locale-config>
  <default-locale>en</default-locale>
  <supported-locale>fr</supported-locale>
</locale-config>
```

These locales should match the extensions of the files (e.g. _en.properties and _fr.properties)

Note: These are the default if the browser does NOT specify any culture/language (based on the browser's settings)

So typically, these values are not as important

In code

Use the FacesContext object to set the locale:

```
Locale french = new Locale(“fr”)  // fr
 or
Locale canadianFrench= new Locale(“fr”,”ca”) // fr-CA
 FacesContext.getCurrentInstance()
             .getViewRoot().setLocale(french);
```

FacesContext object

A new FacesContext object is created with each request

This is the object that stores the component tree we talked about last class

In practice

-Create a managed bean that has a property locale with a getter / setter

-Wire the xhtml to connect with this bean (using a select box most likely)

-Create a *value change event listener* that retrieves the value of the locale and uses it to set the culture

xhtml

```
<h:form>
      <h3><h:outputText value = "#{msg['greeting']}" /></h3>
      <h:panelGrid columns = "2">
        Language :
        <h:selectOneMenu value = "#{userData.locale}"
onchange = "submit()"
          valueChangeListener = "#{userData.localeChanged}">
          <f:selectItems value = "#{userData.countries}" />
        </h:selectOneMenu>
      </h:panelGrid>

    </h:form>
```

<h:selectOneMenu>

-JSF tag used for select drop down box

-Within it, you can add either <f:selectItems> or a list of <f:selectItem>

Example:

```
 <h:selectOneMenu value = "#{userData.data}">
   <f:selectItem itemValue = "1" itemLabel = "Item 1" />
   <f:selectItem itemValue = "2" itemLabel = "Item 2" />
</h:selectOneMenu>
```

-The *itemLabel* is what the client will see. The *itemValue* is how it will show to the server

-The *value* is what data will be bound. For example, when the form is submitted, what bean will get set (e.g. to “1” or “2” above)

<f:selectItems>

If you already have a List or Collection in memory, you can use <f:selectItems>

```
<h:selectOneMenu value = "#{userData.data}">
   <f:selectItems value=“#{userData.options}” />
</h:selectOneMenu>
```

In this case, it will get all the elements in options. If options is a collection, both the itemLabel/itemValue will be the individual elements in the collection (with toString() called on them if necessary to translate to html)

<f:selectItems>

```
<h:selectOneMenu value = "#{userData.data}">
   <f:selectItems value="#{userData.countries}" />
</h:selectOneMenu>
```

If the selectItems is wired to a map of some sort (e.g. HashMap), then the itemLabel will be the *key* from the collection and the itemValue will be the value from the collection. Note that the value can be types other than Strings!

valueChangeListener

In general, on any user input field, you can add a valueChangeListener

This code will fire any time that the component tree is updated with a new value for that UI element.

Remember, that this only happens when the form is submitted or alternatively if an ajax call is made to simulate a form being submitted!

The xhtml

```
<h:selectOneMenu value = "#{userData.locale}" onchange
= "submit()"  valueChangeListener =
"#{userData.localeChanged}">
```

onchange=submit() makes the form submit immediately (same idea as using ajax. Note that this causes the page to reload so it’s simpler but not quite as performant)

localeChanged() is a method within the userData bean (satisfying a specific interface)

Java code

The Java code needs to look like:

```
 public void localeChanged(ValueChangeEvent e) {

  }
```

A ValueChangeEvent object contains information about the value being changed

ValueChangeEvent

A ValueChangeEvent object contains information about the value being changed

The most useful methods are:

getNewValue()

getOldValue()

These both return objects, so you will need to case based on the types of the objects being mapped to

Example for locale
(src: https://www.tutorialspoint.com/jsf/jsf_internationalization.htm)

```
@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {
  private static final long serialVersionUID = 1L;
  private String locale;

  private static Map<String,Object> countries;
    static {

    countries = new LinkedHashMap<String,Object>();
    countries.put("English", Locale.ENGLISH);    // this needs to match the _en.properties
    countries.put("French", Locale.FRENCH); // this needs to match the _fr.properties
  }

  public Map<String, Object> getCountries() { return countries;    }

  public String getLocale() {      return locale; }

  public void setLocale(String locale) {    this.locale = locale;   }

  //value change event listener
  public void localeChanged(ValueChangeEvent e) {
    String newLocaleValue = e.getNewValue().toString();

    for (Map.Entry<String, Object> entry : countries.entrySet()) {

      if(entry.getValue().toString().equals(newLocaleValue)) {
        FacesContext.getCurrentInstance()
          .getViewRoot().setLocale((Locale)entry.getValue());
      }
    }
  }
}
```

Exercise

-Create a resource bundle with 2 string literals for a specific page.

-Test it out by referring to the resource bundle in a JSF page

-Generalize this to use multi language support:

  -Add a variation on the resource (with _fr.properties instead of just .properties)

  -Add a drop down box to allow for this to be changed from within the page

Coming up:

- Running in Waldo instead of local
  - Warning: Sample projects may end up broken after this weekend!
  - Please rename your war file (edit the pom.xml file) that is created to match your team name. e.g. teamAw2019
- How to upload images to your server
- Best practices with password management