INTERNATIONALIZATION

SpringTutors January 10, 2016 Features 81 Views

Internationalization

Internationalization is the process through which application will display the content specific to the locale from where user is accessing the application.

To make any application fully Internationalized we have to work on three areas-

1. Content Internationalization
2. Data base Internationalization
3. NLS programming

When we are storing any data in file or data base we store it with some character encoding like ASCII or UTF-8.

To read the same content from the file or data base we have to read it with same encoding which are used in storing the data otherwise we will get a junk data.

ASCII character set use 8 bytes of character set to display the character.

That is total 256 character can be displayed.

When our data is saved with ASCII character set then our data reader application has also to use ASCII character set to read it.

We know that data are stored in binary format and ASCII character set reads the data by grouping 8 bits.

Let's see how to read 5 in ASCII-

Binary form of 5 is 101

It is stored in database as bits- 00000101

Now to read the data through ASCII it will group the 8 bits together and then form the corresponding character.UTF-8 is also a character set encoding it stores and read the data by grouping 16 bits. This character set will store 32768 bits so it will support maximum forms of data. Today UTF-8 is used very frequently.

NLS programming is known as native language support programming.

To read the data specific to locale we have to develop NLS programs for reading that data. NLS program use the same character set for reading the data which is used in storing the data.

Here we are just doing content internationalization.

Internationalization is supported by J2EE programming.

But there are lots of problem with it.

Let's discuss the internationalization in J2EE.

Let's suppose we have to display "Welcome to Internationalization" in different languages specific to locale.

To display the message we cannot hard code it in JSP instead of we create the properties file for different locale and store the message in it as a key and their respective language format message as a value.

Let's suppose

MessageBundle.properties

Greeting=Welcome to java internationalization

MessageBundle_fr_FR.properties

Greeting=Bienvenue à l'internationalisation java

JSP :-

```
<html>
  <head/>
_<body>
  <%
 InputStream is=new FileInputStream(new File("path where file is stored"));
  Properties props=new Properties();
  Props.load(is);
 String message=Props.getProperty("key");
  System.out.println(message);
  %>
  </body>
```

In the above program we have to write the logic for reading the file and storing it in properties collection and then we will retrieve the message according to locale.

But we do not have to write all the logic because J2EE has provided a class called ResourceBundle which will read the properties file and store it in properties collection based on base name and the locale we have specified.

To read the MessageBundle_fr_FR.properties file we have to pass the base name of properties file that is MessageBundle and we have to set the locale as fr_Fr then it will read the message in French.

```
<%
 ResourceBundle bundle=ResourceBundle.getBundle("MessageBundle",Locale.France);
  System.out.println(bundle.getString("Greeting"));
  %>
```

O/P

Bienvenue à l'internationalisation java

getBundle("baseName",locale) when we pass the baseName and locale it goes to the properties file respective to the locale if it is not found then it will throws an exception.

In the above application we have to write the servlet logic in JSP page it means JSP programmers have to

know the servlet programming also it seems to be difficult.

To solve this problem we should not write the servlet code in JSP to use internationalization, we should write a servlet class and write the logic in service method for reading the properties file specific to the locale and make it as properties collection form and pass it to the JSP page. To get the request and process it in JSP page using JSTL tag

<c:out>.
But now in an application we above to write many servlet which will do the same then we have to duplicate the logic in all the servlet which will create big problems like if any changes made in the properties file names then we have to modify the entire logic in each and every servlet. It creates big maintenance problem.

To solve the above problem we can use Filter. Filter is used to perform preprocessing and postprocessing operation on resource like HTML or JSP or Servlet.

Now we write the logic for reading the properties file specific to the locale using ResourceBundle class in doFilter() method and then forward to the resource(JSP).

When request comes first doFilter() method of Filter is invoked then according to the locale of the client(comes with browser) Filter will perform the operation and forward the request to resource.

Now by applying the above approach we can eliminate the duplication of code among servlets.
Now we can use session object and CacheFilter to make the filter more efficient.

Now we have to create one ResouceBundle object for reading one properties file for one locale. It means for every properties file specified to the locale we have to create ResourceBundle. It is the biggest problem now.

To solve this problem Spring gives ResourceBundleMessageSource class which will read the entire properties file at once and stores it in properties collection.

By using ResourceBundleMessageSource for each and every properties file we not need to create separate ResourceBundleMessageSource, one ResorceBundleMessageSource will read the entire properties file.

We have to configure the ResourceBundleMessageSource as a bean in spring bean configuration file. We have to configure like following-

```
<bean id="resourceBundleMessageSource" class="ResourceBundleMessageSource">
  <property name="basename/basenames">
  <list value-type="java.lang.String">
  <value>MessageBundle</value>
```

There are two properties in ResourceBundleMessageSource class basename(for reading one property file) and basenames(for reading multiple property file) which are of type List<String>.

We will pass the base name of the properties file as a value.

Let's see an example-

Properties files-

MessageBundle.properties

```
Greeting=Hello Jhon
  MessageBundle_fr_FR.properties
  Greeting=Hello Jhon(french)
  MessageBundle_ja_JP.properties
  Greeting=Hello Jhon(Japnese)
```

Spring bean configuration file:

```
<bean id="resourceBundleMessageSource" class="ResourceBundleMessageSource">
  <property name="basenames">
  <list value-type="java.lang.String">
  <value>MessageBundle</value>
  </list>
  </property>
  </bean>
```

```
class Test
  {
  public static void main (String[] args)
  {
  BeanFactory factory=new XmlBeanFactory(new ClassPathResource("application-context.xml"));
 ResourceBundleMessageSource bundle=(ResourceBundleMessageSource)factory.getBean("resourceBundleMessageSource);
  String message=bundle.getMessage("Greeting",null,Locale.FRANCE);
  System.out.println(message);
  }
  }
```

O/P

Hello Jhon(French)

getMessage("String key",object[], locale) method will take three

parameter based on key and locale it will gives us out put if we use locale as JAPAN in above getMessage() method then it will gives

Hello Jhon(Japnese) as O/P

Now we can see that for different Properties file specific to the locale we have not to create multiple ResourceBundleMessageSource object only one object is required.

Now if any changes made in the properties file while running the application then ResourceBundleMessageSource will not show that changes because it will read the file in starting and does not reloade it while running the application. This is the problem with ResourceBundleMessageSource.

To solve the above problem spring has given one more class called ReloadableResourceBundleMessageSource. It will reload the properties file at every time interval specified in it. So by using this run time changes made in properties file will also be seen.

We can configure the ReloadableResourceBundleMessageSource as-

```
<bean id="reloadableResourceBundleMessageMessageSource" class=" ReloadableResourceBundleMessageMessageSource ">
  <property name="basenames">
  <list value-type="java.lang.String">
  <value>MessageBundle</value>
  </list>
  </property>
  </bean>
```

Both ResourceBundleMessageSource And ReloadableResourceMessageSource are the implementation class of MessageSource interface.
Now when we are using any of the implementation class name of the MessageSource in our code then we get tightly coupled with spring.
To avoid this we can use interface MessageSource like following

```
MessageSource message=factory.getBean("resourceBundleMessageSource/reloadableResourceBundleMessageSource",MessageSource.class);
```

We know base class reference can hold the object of derived class.
Now if we use MessageSource interface in our class then spring will violate its one of the biggest principle called non-invasive.
Hence we can see that BeanFactory will not support the internationalization.

In spring we can create IOC container in two ways-

1. BeanFactory
2. ApplicationContext

BeanFactory

BeanFactory factory=new XmlBeanFactory(new ClassPathResource("application-context.xml"));
When we create IOC container with BeanFactory first ClassPathResource goes to the physical Xml bean configuration file it reads that file and store it in resource object and gives it to XmlBeanFactory.
Now XmlBeanFactory will check the xml file is welformed or not then it check its validation if both are passed then it will create an empty IOC container and place the xml file as in memory metadata.It will starts creating the object of bean when the bean is requested.
That's why it is called as lazy initializer.

ApplicationContext

ApplicationContext context=new ClassPathXmlApplicationContext("application-context.xml");

When we create the IOC container with ApplicationContext first ClassPathXmlApplicationContext will go to the physical xml bean configuration file reads it and load it by creating IOC container as in memory metadata. Now ApplicationContext will read the bean definition one by one and creates the object for the bean whose scopes are singileton. It means before referring to the bean whose scopes are singileton it will create the object.

That's why it is called as eagerly initializer.

BeanFactory VS ApplicationContext

As ApplicationContext creates the object while creating the IOC container it will go through all the bean definition so once it creates the object it will not throw exception at run-time.

While BeanFactory will only checks welformness and validation.

As it does not create the object while creating the IOC container it does not go through the bean definition without any request so when we request for bean object then it will reads the bean definition and creates the object if bean definition has some error then it will throw run-time exception.

It means ApplicationContext IOC container gives prof that it will not throw exception while running. If any error exist in bean definition then it will not create the IOC container.

Spring will support the internationalization through ApplicationContext. It contains the ResourceBundleMessageSource and ReloadableResourceBundleMessageSource.

Spring has provided method called getMessage(—) which will read the message through ApplicationContext like following-

```
Application context=new ClassPathXmlApplicationContext("application-context.xml");
  String message=context.getMessage("key"objet[],locale);
  System.out.println(message);
```

When we invoke getMessage() method ApplicationContext will search ResourceBundleMessageSource in IOC container if it finds then read the message from properties file specific to the locale and returns it to us.

But question is that how ApplicationContext will find ResourceBundleMessageSource in IOC container because we can configure the ResourceBundleMessageSource many times in spring bean configuration file.

To identify the ResourceBundleMessageSource which will read the properties file in IOC container Application context has given an attribute called "messageSource".

We have to configure the ResourceBundleMessageSource id as "messageSource" only.

For the above application we have to configure the spring bean configuration file for ApplicationContext as

follows

```
<bean id="messageSource" class="ResourceBundleMessageSource">
  <property name="basenames">
  <list value-type="java.lang.String">
  <value>MessageBundle</value>
  <list>
  <property>
  </bean>
```

Place holder

Now we can use place holder in properties file like following:

Error.properties

```
firstName={0} should cannot be empty
 ApplicationContext context=new ClassPathXmlApplicationContext("application-context.xml");
   context.getMessage("firstname",new object[]{"First Name"},Locale.getDefault());
```

O/P

First Name cannot be empty

We can initialize the place holder at run time through object array.