Capítulo 14: Internacionalización en JavaFX

Examples

Cargando paquete de recursos

JavaFX proporciona una manera fácil de internacionalizar sus interfaces de usuario. Al crear una vista desde un archivo FXML, puede proporcionar al `FXMLLoader` un paquete de recursos:

```
Locale locale = new Locale("en", "UK");
ResourceBundle bundle = ResourceBundle.getBundle("strings", locale);

Parent root = FXMLLoader.load(getClass().getClassLoader()
                                  .getResource("ui/main.fxml"), bundle);
```

Este paquete proporcionado se usa automáticamente para traducir todos los textos en su archivo FXML que comienzan con un `%` . Digamos que el archivo de propiedades `strings_en_UK.properties` contiene la siguiente línea:

```
ui.button.text=I'm a Button
```

Si tienes una definición de botón en tu FXML como esta:

```
<Button text="%ui.button.text"/>
```

Recibirá automáticamente la traducción de la clave `ui.button.text` .

Controlador

A Los paquetes de recursos contienen objetos específicos del entorno local. Puede pasar el paquete al `FXMLLoader` durante su creación. El controlador debe implementar la interfaz `Initializable` y anular el método de `initialize(URL location, ResourceBundle resources)` . El segundo parámetro de este método es `ResourceBundle` que se pasa del FXMLLoader al controlador y puede ser utilizado por el controlador para traducir textos adicionales o modificar otra información dependiente de la ubicación.

```
public class MyController implements Initializable {

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        label.setText(resources.getString("country"));
    }
}
```

Cambio dinámico de idioma cuando la aplicación se está ejecutando

Este ejemplo muestra cómo construir una aplicación JavaFX, donde el idioma se puede cambiar

dinámicamente mientras la aplicación se está ejecutando.

Estos son los archivos de paquete de mensajes utilizados en el ejemplo:

messages_en.properties :

```
window.title=Dynamic language change
button.english=English
button.german=German
label.numSwitches=Number of language switches: {0}
```

messages_de.properties :

```
window.title=Dynamischer Sprachwechsel
button.english=Englisch
button.german=Deutsch
label.numSwitches=Anzahl Sprachwechsel: {0}
```

La idea básica es tener una clase de utilidad I18N (como alternativa, esto podría implementarse un singleton).

```
import javafx.beans.binding.Bindings;
import javafx.beans.binding.StringBinding;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Locale;
import java.util.ResourceBundle;
import java.util.concurrent.Callable;

/**
 * I18N utility class..
 */
public final class I18N {

    /** the current selected Locale. */
    private static final ObjectProperty<Locale> locale;

    static {
        locale = new SimpleObjectProperty<>(getDefaultLocale());
        locale.addListener((observable, oldValue, newValue) -> Locale.setDefault(newValue));
    }

    /**
     * get the supported Locales.
     *
     * @return List of Locale objects.
     */
    public static List<Locale> getSupportedLocales() {
        return new ArrayList<>(Arrays.asList(Locale.ENGLISH, Locale.GERMAN));
    }
```

```
    /**
     * get the default locale. This is the systems default if contained in the supported
locales, english otherwise.
     *
     * @return
     */
    public static Locale getDefaultLocale() {
        Locale sysDefault = Locale.getDefault();
        return getSupportedLocales().contains(sysDefault) ? sysDefault : Locale.ENGLISH;
    }

    public static Locale getLocale() {
        return locale.get();
    }

    public static void setLocale(Locale locale) {
        localeProperty().set(locale);
        Locale.setDefault(locale);
    }

    public static ObjectProperty<Locale> localeProperty() {
        return locale;
    }

    /**
     * gets the string with the given key from the resource bundle for the current locale and
uses it as first argument
     * to MessageFormat.format, passing in the optional args and returning the result.
     *
     * @param key
     *         message key
     * @param args
     *         optional arguments for the message
     * @return localized formatted string
     */
    public static String get(final String key, final Object... args) {
        ResourceBundle bundle = ResourceBundle.getBundle("messages", getLocale());
        return MessageFormat.format(bundle.getString(key), args);
    }

    /**
     * creates a String binding to a localized String for the given message bundle key
     *
     * @param key
     *         key
     * @return String binding
     */
    public static StringBinding createStringBinding(final String key, Object... args) {
        return Bindings.createStringBinding(() -> get(key, args), locale);
    }

    /**
     * creates a String Binding to a localized String that is computed by calling the given
func
     *
     * @param func
     *         function called on every change
     * @return StringBinding
     */
    public static StringBinding createStringBinding(Callable<String> func) {
```

```
        return Bindings.createStringBinding(func, locale);
    }

    /**
     * creates a bound Label whose value is computed on language change.
     *
     * @param func
     *         the function to compute the value
     * @return Label
     */
    public static Label labelForValue(Callable<String> func) {
        Label label = new Label();
        label.textProperty().bind(createStringBinding(func));
        return label;
    }

    /**
     * creates a bound Button for the given resourcebundle key
     *
     * @param key
     *         ResourceBundle key
     * @param args
     *         optional arguments for the message
     * @return Button
     */
    public static Button buttonForKey(final String key, final Object... args) {
        Button button = new Button();
        button.textProperty().bind(createStringBinding(key, args));
        return button;
    }
}
```

Esta clase tiene una `locale` campo estático que es un objeto de `locale` `Locale` Java envuelto en una `ObjectProperty` objeto JavaFX, por lo que se pueden crear enlaces para esta propiedad. Los primeros métodos son los métodos estándar para obtener y establecer una propiedad JavaFX.

La `get(final String key, final Object... args)` es el método central que se utiliza para la extracción real de un mensaje de un `ResourceBundle` .

Los dos métodos llamados `createStringBinding` crean un `StringBinding` que está vinculado al campo de `locale` y, por lo tanto, los enlaces cambiarán cada vez que cambie la propiedad de la `locale` . El primero usa sus argumentos para recuperar y formatear un mensaje usando el método de `get` mencionado anteriormente, el segundo se pasa en un valor de `Callable` , que debe producir el nuevo valor de cadena.

Los dos últimos métodos son métodos para crear componentes JavaFX. El primer método se utiliza para crear una `Label` y utiliza un `Callable` para su enlace de cadena interno. El segundo crea un `Button` y utiliza un valor de clave para la recuperación del enlace de cadena.

Por supuesto, se podrían crear muchos más objetos diferentes como `MenuItem` o `ToolTip` pero estos dos deberían ser suficientes para un ejemplo.

Este código muestra cómo se usa esta clase dentro de la aplicación:

```
import javafx.application.Application;
```

```
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

import java.util.Locale;

/**
 * Sample application showing dynamic language switching,
 */
public class I18nApplication extends Application {

    /** number of language switches. */
    private Integer numSwitches = 0;

    @Override
    public void start(Stage primaryStage) throws Exception {

        primaryStage.titleProperty().bind(I18N.createStringBinding("window.title"));

        // create content
        BorderPane content = new BorderPane();

        // at the top two buttons
        HBox hbox = new HBox();
        hbox.setPadding(new Insets(5, 5, 5, 5));
        hbox.setSpacing(5);

        Button buttonEnglish = I18N.buttonForKey("button.english");
        buttonEnglish.setOnAction((evt) -> switchLanguage(Locale.ENGLISH));
        hbox.getChildren().add(buttonEnglish);

        Button buttonGerman = I18N.buttonForKey("button.german");
        buttonGerman.setOnAction((evt) -> switchLanguage(Locale.GERMAN));
        hbox.getChildren().add(buttonGerman);

        content.setTop(hbox);

        // a label to display the number of changes, recalculating the text on every change
        final Label label = I18N.labelForValue(() -> I18N.get("label.numSwitches",
numSwitches));
        content.setBottom(label);

        primaryStage.setScene(new Scene(content, 400, 200));
        primaryStage.show();
    }

    /**
     * sets the given Locale in the I18N class and keeps count of the number of switches.
     *
     * @param locale
     *         the new local to set
     */
    private void switchLanguage(Locale locale) {
        numSwitches++;
        I18N.setLocale(locale);
    }
}
```

La aplicación muestra tres formas diferentes de usar el `StringBinding` creado por la clase `I18N` :

1. el título de la ventana está enlazado directamente mediante un `StringBinding` .
2. Los botones utilizan el método auxiliar con las teclas de mensaje.
3. la etiqueta utiliza el método de ayuda con un `Callable` . Este `Callable` utiliza el método `I18N.get()` para obtener una cadena traducida formateada que contiene el recuento real de conmutadores.

Al hacer clic en un botón, se incrementa el contador y se establece la propiedad de configuración regional de `I18N` , lo que a su vez desencadena el cambio de enlaces de cadenas y, por lo tanto, establece la cadena de la interfaz de usuario en nuevos valores.

Lea Internacionalización en JavaFX en línea:
https://riptutorial.com/es/javafx/topic/5434/internacionalizacion-en-javafx