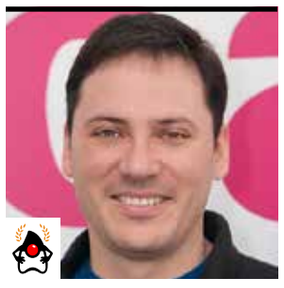

ANDRÉS ALMIRAY

FXMLLoaderによるFXMLの拡張

FXMLLoaderメカニズムを活用し、JavaFX UIの宣言的定義でより高い柔軟性を実現する

JavaFX UIツールキットは、JVMで動作するデスクトップ・アプリケーションおよびモバイル・アプリケーションの構築に利用できる最新のAPIセットを提供しています。JavaFXでは、前身であるSwingと同様に、標準のウィジェット・セットが提供されています。さらに、カスタム・コンポーネントや新たな動作によってウィジェットを拡張する手段も提供されています。

JavaFXには、プロパティ・バインディング、CSSによるスタイル設定、FXMLというUI記述フォーマットなどの新機能も追加されています。名前からわかるように、JavaFXは開発者が宣言的にユーザー・インタフェースを定義できる、XMLベースのフォーマットです。これは、JavaFX APIを直接利用する、手続き型インタフェース定義とは異なります。

プログラミング型APIと比較した場合、FXMLには次のようなメリットがあります。

- FXMLは階層型フォーマットです。JavaFX SceneGraphも、UI要素をツリー型データ構造で表しているため、階層型構造であると言えます。SceneGraphの各ノードは、ボタン、ラベル、テキスト・フィールドなどのグラフィック要素に関連付けられています。そのため、プレーン・コードに比べてFXMLはコンポーネント階層の視覚化が容易です。
- FXMLは必要に応じてすぐに作成できるため、アプリケーションの実行時に特定のタイミングで動的にUI要素の追加や作成を行うことが可能です。
- 多くの場合、FXMLを書く方がUIの定義が短く済みます。

おそらく、皆さんはすでにFXMLを見たことがあると思いますが、そうでない方のために簡単な例を紹介します。次のコード・スニペットでは、6つのUI要素を2列のレイアウトに配置したグリッドを定義しています。

sample/app.fxml

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.GridPane?>
<GridPane>
  <Label text="Username:"
    GridPane.columnIndex="0" GridPane.rowIndex="0"/>
  <TextField
    GridPane.columnIndex="1" GridPane.rowIndex="0"/>
  <Label text="Password:"
    GridPane.columnIndex="0" GridPane.rowIndex="1"/>
  <PasswordField
    GridPane.columnIndex="1" GridPane.rowIndex="1"/>
  <Button text="Cancel"
    GridPane.columnIndex="0" GridPane.rowIndex="2"/>
  <Button text="Login"
    GridPane.columnIndex="1" GridPane.rowIndex="2"/>
</GridPane>
```

図1に、このアプリケーションの実行画面を示します。

Javaコードの場合と同様に、FXMLファイルで型を使用するためには、import文を指定する必要がある点に注意してください。このimport文は、FXMLローディング・メカニズムに対するヒントとなり、記述された宣言的UIの解析や、UI要素が含まれる適切なSceneGraphへの変換に利用されます。このローディング・メカニズムは、FXMLLoaderと呼ばれています。FXMLLoaderは簡単に利用できます。次のコードをご覧ください。

sample/App.java

```
package sample;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.net.URL;

public class App extends Application{
   @Override
   public void start(Stage stage) throws Exception {
      URL fxml = getClass().getClassLoader()
            .getResource("sample/app.fxml");
      FXMLLoader fxmlLoader = new FXMLLoader(fxml);

      stage.setScene(new Scene(fxmlLoader.load()));
      stage.sizeToScene();
      stage.show();
   }
}
```

この段階でFXMLLoaderに渡す必要があるのは、ロードするFXMLリソースの場所のみです。ただし、UI要素を作成してSceneGraphにアタッチするアクションは、UIスレッド（FXアプリケーション・スレッド）内で実行しなければならない点を覚えておいてください。このルールに違反した場合、厄介なことが発生する可能性があります。ただし、基本型であるApplicationでは、コードをアプリケーション・スレッドで実行する基本ライフサイクルが提供されています。今回の例のstartメソッドは、UIスレッドで呼び出されることが保証されているため、何の問題も起こりません。

ノードのプロパティ

最初の例のFXMLスニペットをじっくり見てみると、LabelおよびButtonというFXMLノードにtext属性が定義されていることがわかります。この属性は、対応する型のJavaBeanプロパティにマッピングされます。そのため、FXMLLoaderは、最初に見つかったLabelをインスタンス化する際に、そのラベルのtextプロパティにこのtext属性の値を正確に設定します。つまりこの動作は、FXMLLoaderが自動的に次のコードを呼び出しているように見えます。

```
Label label = new Label();
label.setText("Username:");
// ラベルをSceneGraphに挿入
```

このスニペットはとても短いものであり、現時点で機能的には大したメリットはないように思われるかもしれません。しかし、UI要素には、複数のプロパティを設定しなければならない場合があることを考えてみてください。定義するノードや属性が増えれば増えるほど、FXMLのメリットは明らかになります。なお、ノード型（この場合はLabel）には、JavaBeansの表記規則に従ってプロパティを定義しなければなりません。そうすることによって、プロパティの問合せ（getter）と変更（setter）に使える2つのメソッドが生成されます。いずれのメソッドもない場

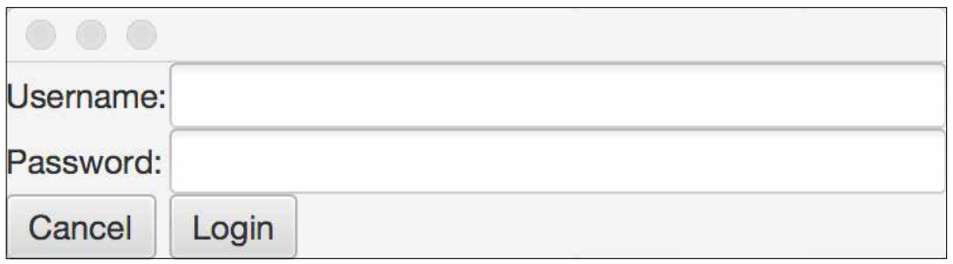

図1：FXMLによるログイン画面

合、FXMLLoaderからプロパティを参照することはできません。特定のFXMLスニペットが予想どおりには動作しない場合、この点が、その理由を突き止める最初の手がかりになります。つまり、設定しようとしているプロパティがJavaBeansプロパティとして正しく定義されていない可能性があります。

また、先ほどの6つの要素すべてには、GridPaneを名前空間として使うように指定する必要があるプロパティが追加されていますが、その意味がよくわからない方もいらっしゃるかもしれません。これらのプロパティは、ターゲット・ノード自体のプロパティではありませんが、ノードに影響を与えています。ノードへの影響は、columnIndexやrowIndexの値を変更することにより確認できます。変更後の値によっては、奇妙なレイアウトになることがあります。

それでは、ここで何が起きているのでしょうか。結論を言えば、このプロパティは、ターゲット・ノードではなく、GridPaneクラスで定義されています。名前空間としてクラス名が使われているのはそのためです。今回利用している2つのプロパティは、GridPaneクラスの静的メソッドとして定義されています。つまり、これらのプロパティはJavaBeansプロパティそのものではなく、FXMLがサポートする表記規則の一部です。先ほどと同じく、各プロパティにgetterとsetterのペアとなるメソッドが必要です。また、各メソッドにはstatic修飾子が必要です。さらに、この表記規則を使う場合、メソッドの最初の引数がターゲット・ノードである必要があります。GridPaneには、以上の規則に従った次のメソッドが定義されています。

```
public static void
    setColumnIndex(Node node, int index);
public static int getColumnIndex(Node node);
public static void setRowIndex(Node node, int index);
public static int getRowIndex(Node node);
```

最初の引数には、JavaFXのUI要素すべてのスーパータイプであるNode型を利用できます。標準のウィジェット・セットの一部であるか、サードパーティ製ライブラリの一部であるかを問わず、すべての種類のUI要素に対してGridPaneを利用できるのはそのためです。先ほどのメソッド定義では、2番目の引数として整数型が指定されていることにお気付きと思いますが、FXMLファイルで対応する値はリテラルとして定義されています。これがうまく動作するのは、FXMLLoaderには型変換機能があり、すべての基本型および列挙型を変換できるためです。もう1つ気になる点は、指定されたノードに関連する値をどのように保存しているのかでしょう。JavaFX APIはこの点もサポートしてくれます。基本型のNodeには便利なプロパティ・マップがあり、任意の種類の値を保存するために使用できるようになっています。実は、先ほどのメソッドはこのマップに値を書き込んでいます。

以上の新知識を活用すれば、独自の合成プロパティを作成できます。合成プロパティとは、ターゲット型の内部で明示的には定義されていないプロパティです。
合成プロパティは、コンテキスト依存ディクショナリなどの外部的な手段によってターゲット型に関連付けられます。例として、Usernameフィールドまたはpasswordフィールドにユーザーが入力できる文字数を制限する必要がある場合を考えてみます。その場合、Nodeと、制限する文字数を指定する数値を受け取る1対のメソッドを定義した新しいユーティリティ・クラスを作成できます。このユーティリティ・クラスは、次のようになります。

```
package sample;

import javafx.scene.control.TextInputControl;

public final class InputControlUtils {
  private static final String LIMIT = " limit";

  public static void setMaxTextLimit(
          TextInputControl control, int maxLength) {
   control.getProperties().put(LIMIT, maxLength);
   control.textProperty()
    .addListener((ov, oldValue, newValue) -> {
     if (control.getText().length() > maxLength) {
       String s = control.getText()
               .substring(0, maxLength);
       control.setText(s);
     }
    });
  }
```

```
    public static int getMaxTextLimit(
                        TextInputControl control) {
        Object value =
            control.getProperties().get(LIMIT);
        if (value instanceof Number) {
            return ((Number) value).intValue();
        }
        return -1;
    }
}
```

なお、このコードでは無効な値（maxLengthが0以下の場合など）を考慮しておらず、同じターゲット・ノードに対してsetMaxTestLimitメソッドが複数回呼ばれた場合にリスナーの登録を解除する手段も提供していない点に注意してください。このコードは例示を目的としたものであるため、追加の調整なしにそのまま本番環境で利用すべきではありません。

次に、FXML定義を次のように更新すればよいだけです。

sample/app.fxml

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.GridPane?>
<?import sample.InputControlUtils?>
<GridPane>
  <Label text="Username:"
    GridPane.columnIndex="0" GridPane.rowIndex="0"/>
  <TextField
    GridPane.columnIndex="1" GridPane.rowIndex="0"
    InputControlUtils.maxTextLimit="10"/>
  <Label text="Password:"
    GridPane.columnIndex="0" GridPane.rowIndex="1"/>
  <PasswordField
    GridPane.columnIndex="1" GridPane.rowIndex="1"
    InputControlUtils.maxTextLimit="10"/>
  <Button text="Cancel"
    GridPane.columnIndex="0" GridPane.rowIndex="2"/>
  <Button text="Login"
    GridPane.columnIndex="1" GridPane.rowIndex="2"/>
</GridPane>
```

FXMLでプロパティを扱う際の機能について、さらに2つのことを紹介しておきます。1つ目は、デフォルト・プロパティの名前を定義できることです。これによって、階層を定義する際にプロパティ名を省略できるようになります。そのプロパティ名は、コードでターゲット型に@javafx.beans.DefaultPropertyアノテーションを適用して定義する必要があります。標準のJavaFXウィジェット・セットでは、多くのUI要素でこのアノテーションが使われています。たとえば、さまざまなウィジェット・コンテナの基本型であるjavafx.scene.layout.Pane型には、@DefaultProperty("children")というアノテーションが付加されています。これによって、次のFXMLの代わりに、先ほどの例のような記述が可能になります。

sample/app.fxml

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.GridPane?>
<GridPane>
  <children>
    <Label text="Username:"
      GridPane.columnIndex="0" GridPane.rowIndex="0"/>
    <!-- その他の要素 -->
  </children>
</GridPane>
```

よく使われているIDEでは、FXMLのサポートが高度に統合されているため、FXMLファイルで記述する必要のないデフォルト・プロパティがある

かどうかを指摘させることができます。なお、FXMLは手で書くこともできますが、無償のビジュアル・ツールであるScene Builderを使うこともできるのは注目に値します。このツールは、本号12ページで特集しています。

また、型をインスタンス化する際に、FXMLLoaderに対して追加のヒントを提供できることも紹介しておくべきでしょう。今までの例では、JavaBeansの表記規則に従った、コンストラクタに引数を指定していない型のみを紹介してきました。FXMLLoaderがデフォルトで想定しているのは、そのような型です。しかし、引数を取るコンストラクタが定義されているノードを扱うこともあるかもしれません。そのようなノードで追加のヒントを提供できなかった場合、FXMLLoaderはノードのインスタンス作成に失敗します。追加のヒントは、通常のプロパティと同じように、コンストラクタの引数名を指定して提供することができます。その際に使用するのが、@javafx.beans.NamedArgアノテーションです。引数名の定義は必須です。FXMLファイルで属性を省略している場合は、引数のデフォルト値を定義できます。具体的な例を挙げます。標準JavaFX APIのInsetsクラスでは、次のコンストラクタが定義されています。

```
public Insets(@NamedArg("top") double top,
        @NamedArg("right") double right,
        @NamedArg("bottom") double bottom,
        @NamedArg("left") double left) {
    this.top = top;
    this.right = right;
    this.bottom = bottom;
    this.left = left;
}

public Insets(@NamedArg("topRightBottomLeft")
        double topRightBottomLeft) {
    this.top = topRightBottomLeft;
    this.right = topRightBottomLeft;
    this.bottom = topRightBottomLeft;
    this.left = topRightBottomLeft;
}
```

このようにアノテーションを使うと、次に示すいずれの方法でもインスタンスを定義できます。

```
<Insets top="10" right="10" bottom="10" left="10"/>
```

または

```
<Insets topRightBottomLeft="10"/>
```

今のところ、FXMLでサポートされているプロパティについては以上のことを知っておけば十分でしょう。次は、国際化を例に、FXMLLoader自体を設定する方法について見てゆきます。

FXMLLoaderの国際化設定

現在、世界ではさまざまな言語が話されています。開発者は、作成するアプリケーションのユーザーを考慮しなければなりません。ユーザーの中には、開発者と異なる言語を話す人もいます。そのため、ユーザーが必要とする言語を組み込んだバージョンを提供する用意をしなければなりません。

JavaFXウィジェットは、JVMから知らされる、ロケールなどの地域設定に対応しています。そのため、ウィジェットは状況に応じて、右から左に描画される場合と、左から右に描画される場合があります。しかし、開発者がアプリケーションに追加する文字列リテラルは自動変換されません。それらのリテラルを何らかの方法で外部に切り出してFXML
ファイルから参照することができれば、それに越したことはありません。FXMLLoaderでは、この問題を解決する方法
が提供されています。具体的には、ローダーの生成時にResource Bundleを渡すか、FXMLファイルを読み込む前にこのバンドルをプロパティとして提供することができます。このResourceBundleは、任意の方法で作成できます。今回の例のsample/App.javaは、次のように更新できます。

```
ClassLoader cl = getClass().getClassLoader();
URL fxml = cl.getResource("sample/app.fxml");
String resource = "sample/messages_en.properties"
```

```
InputStream is =
   cl.getResourceAsStream(resource);
ResourceBundle bundle =
   new PropertyResourceBundle(is);
FXMLLoader fxmlLoader = new FXMLLoader(fxml, bundle);
```

さらに、デフォルトのロケールとして英語（en）バージョンのプロパティ・ファイルを、それとは別にドイツ語（de）バージョンのプロパティ・ファイルを定義します。

sample/messages_en.propertiesファイルは次のようにします。

```
username.label=Username:
password.label=Password:
action.cancel.label=Cancel
action.login.label=Login
```

sample/messages_de.propertiesファイルは次のようにします。

```
username.label=Benutzername:
password.label=Passwort:
action.cancel.label=Abbrechen
action.login.label=Anmeldung
```

定義したプロパティのキーを反映させるために、FXMLファイルも変更する必要があります。キーの値とリテラル値を区別するために、%を接頭辞として使用します。IDEによっては、キー補完を使用できることもあります。

```
<Label text="%username.label"
   GridPane.columnIndex="0" GridPane.rowIndex="0"/>
<TextField
   GridPane.columnIndex="1" GridPane.rowIndex="0"
   InputControlUtils.maxTextLimit="10"/>
<Label text="%password.label"
   GridPane.columnIndex="0" GridPane.rowIndex="1"/>
<PasswordField
   GridPane.columnIndex="1" GridPane.rowIndex="1"
   InputControlUtils.maxTextLimit="10"/>
<Button text="%action.cancel.label"
   GridPane.columnIndex="0" GridPane.rowIndex="2"/>
<Button text="%action.login.label"
   GridPane.columnIndex="1" GridPane.rowIndex="2"/>
```

再度アプリケーションを実行すると、図1と同じ画面が表示されます。ここで、ドイツ語バージョンのプロパティ・ファイルを読み込むようにコードを変更すると、アプリケーションの画面は図2のようになります。

すばらしいですね。これは、ローカライゼーションに対応したアプリケーションを作成する第一歩になります。先ほど説明したように、ResourceBundleは一般的なJavaプログラミングでよく使われるさまざまな方法で作成できます。次のステップは、アプリケーションを実行時のロケール変更に対応させることでしょう。その場合、ロケールの変更が発生し、キーに対応する新しい値を再度取得する必要があることをウィジェットに通知する方法を見つける必要があります。FXMLLoaderだけで実現するのは困難ですが、これを始めるための基本的なパーツはそろっています。

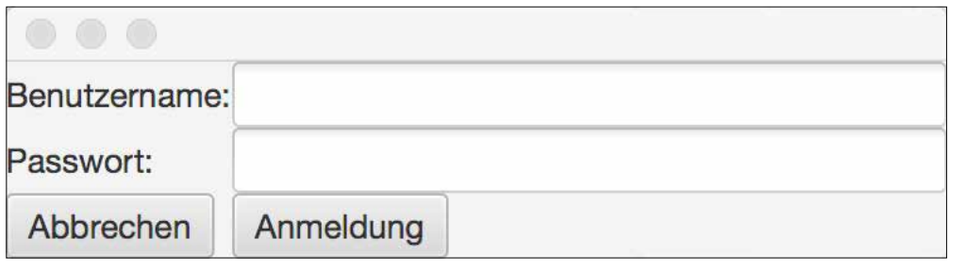

図2：ドイツ語でのログイン画面

まとめ

FXMLを使うと、簡単かつ短時間でUIを宣言的に定義できます。ツール（IDEやSceneBuilder）によるサポートも十分であるため、FXMLを書くにあたってのハードルは低くなっています。FXML内で使用できるノード型の多くはJavaBeans表記規則に従っているため、簡単に使うことができます。表記規則が異なるものであっても、便利な拡張フォーマットが提供されており、ローディング・メカニズムであるFXMLLoaderにヒントを渡すこともできるため、FXMLで利用できます。さらに、FXML経由でウィジェットのローカライゼーションに関する問題も設定できるため、アプリケーションの対象ユーザーを拡大できます。次号の記事では、fx:controller属性や@FXMLアノテーションなどの機能を取り上げ、FXMLファイルで定義したウィジェットの高度なカスタマイズやバインドを可能にする、FXMLLoaderやFXMLのメカニズムについて説明する予定です。</article>

Andrés Almiray：JavaおよびGroovyの開発者で、Java Champion。17年を超えるソフトウェアの設計および開発の経験があり、Javaの初期の頃から、Webアプリケーションやデスクトップ・アプリケーションの開発に関わっている。熱心なオープン・ソースの信奉者で、Groovy、JMatter、Asciidoctorなどの有名プロジェクトに参加。Griffonフレームワークの創設メンバーであり、現在のプロジェクト・リードである。JSR 377の仕様リードも務めている。

learn more

[オラクルによるFXMLに関するチュートリアル]

O'REILLY®

Software Architecture

ENGINEERING THE FUTURE OF SOFTWARE

The brightest minds in tech are coming to New York.

Practical training in the tools, techniques, and leadership skills you need for the evolving world of software architecture.

April 2–5, 2017
softwarearchitecturecon.com/ny

JAVA MAGAZINE READERS GET
20% OFF WITH CODE **JAVA**