JAMES L. WEAVER

BIO

パート2

JavaFX 2における遅延評価と遅延初期化、そしてカスタム・バインディング

バインディングの評価とプロパティの初期化を遅延方式で最適化する

JavaFX 2 は、リッチ・インターネット・アプリケーション（RIA）を作成するためのAPIおよび実行環境です。Java FXは2007年に登場し、2011年10月にバージョン2がリリースされました。JavaFX 2の利点の1つは、すでに成熟している使い慣れたツールを使用してJava言語のコードを作成できる点です。

本記事は、全2回シリーズのパート2です。今回はJavaFX 2のプロパティとバインディングを最適化するための遅延評価、遅延初期化、およびカスタム・バインディングの実装に焦点を置きます。

前号の記事「JavaFX 2.0におけるプロパティとバインディングの活用：パート1」で説明したとおり、JavaFX 2には、**図1**に示した一連のインタフェースが含まれています。これらのインタフェースは、プロパティの使用と実装、プロパティ値の変更の検出、およびプロパティ間のバインディングに関するサポートを提供することを目的としています。

図1の各インタフェースは次の4つのパッケージのいずれかに属しています。

- javafx.beans
- javafx.beans.binding
- javafx.beans.property
- javafx.beans.value

本記事では、これらのインタフェースの多くで定義されているメソッドを使用して、遅延評価、遅延初期化、およびカスタム・バインディングを実装する例を説明します。

LazyInitEvalSolutionアプリケーションの概要

本記事では、プロパティとバインディングの使用法の習得に役立てるため、LazyInitEvalSolutionというサンプル・アプリケーションを使用します。このアプリケーションには、**図2**に示すとおり、3つのストップウォッチがあります。それぞれのストップウォッチには複数のボタンがあり、経過時間とラップ・タイムが表示されます。

次のセクションでダウンロードするLazyInitEvalExerciseプロジェクトには、このサンプル・アプリケーションの初期コードが含まれています。ダウンロードした状態では、アプリケーション実行時の外観は図2のようになります。本記事を進める過程で、LazyInitEvalExerciseプロジェクトのコードを変更し、LazyInitEvalSolutionプロジェクトの遅延評価、遅延初期化、カスタム・バインディングの各動作を実装していきます。LazyInitEvalSolutionプロジェクトもダウンロード・ファイルに含まれています。

図3に示すとおり、いずれかのストップウォッチの「Start」ボタンをクリックした場合、ミリ秒単位で時間を計測する経過タイマーがスタートします。しかし、経過時間を表示するため

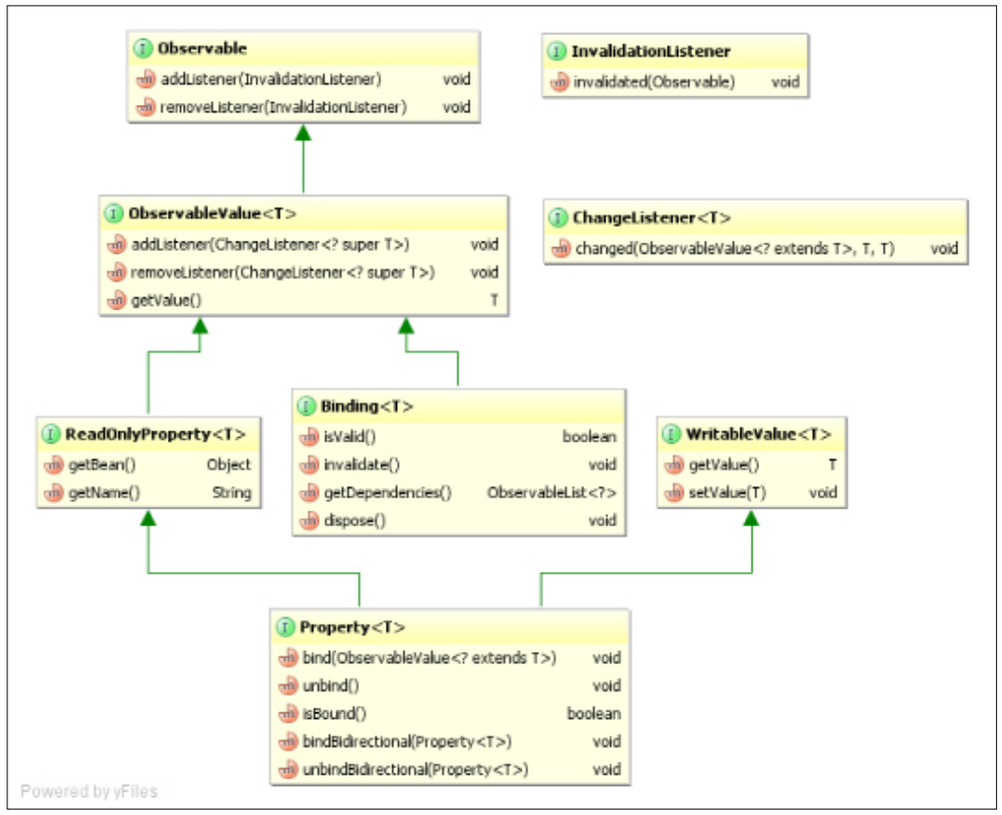

図1

写真:STEVE GRUBMAN

には、「**invalidate - click to bind/**」というテキストをクリックする必要があります。コードを確認していく過程で、これが遅延評価の概念を表していることがわかります。

また、「**Lap**」ボタンをクリックした場合、その時点のラップ・タイムが表示されます。「**Stop**」ボタンをクリックした場合、経過時間の計測が停止します。さらに、「**Reset**」ボタンをクリックした場合は、経過時間とラップ・タイムの両方がリセットされます。

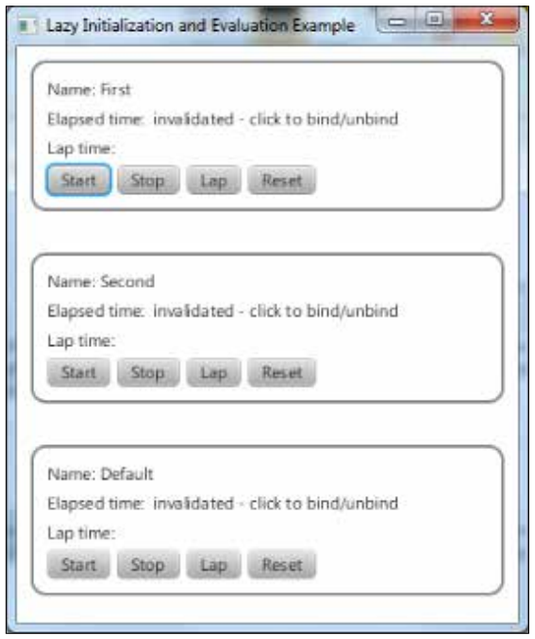

図2

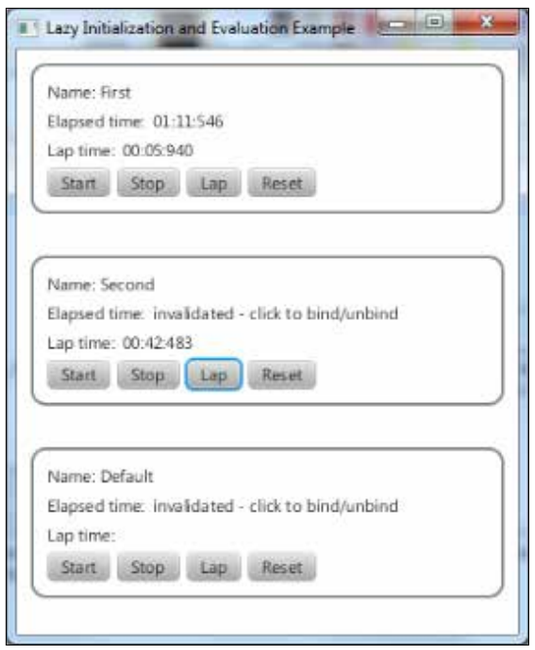

図3

LazyInitEvalExerciseプロジェクトの取得と実行

1. LazyInitEvalExerciseプログラムを含むNetBeansプロジェクト・ファイルをダウンロードします。
2. ダウンロードしたプロジェクトを任意のディレクトリに解凍します。
3. NetBeansを起動し、「**File**」→「**Open Project**」を選択します。
4. 「Open Project」ダイアログ・ボックスで、LazyInitEvalExerciseプロジェクトを解凍したディレクトリに移動し、プロジェクトを開きます（**図4**参照）。jfxrt.jarファイルが見つからないというメッセージが表示された場合は、「**Resolve**」ボタンをクリックし、JavaFX 2 SDKをインストールしたディレクトリの下にあるrt/libフォルダに移動してください。
   注:NetBeans IDEはNetBeansのWebサイトから入手できます。
5. ツールバーの**Run Project**アイコンをクリックするか［F6］キーを押して、アプリケーションを実行します。Run Projectアイコンは、**図5**に示すとおり、DVDプレーヤの再生ボタンに似たアイコンです。

LazyInitEvalExerciseアプリケーションが**図6**のようにウィンドウに表示されます。

LazyInitEvalExerciseの動作は、いくつかの点でLazyInitEvalSolutionと異なります。たとえば、「**Start**」ボタンをクリックし、次に「**invalidated - click to bind/unbind**」テキストをクリックした場合、経過時間は「分:秒:ミリ秒」の形式ではなく整数で表示されます（**図6**参照）。LazyInitEvalSolutionでは、カスタム・バインディングを使用しているため、「分:秒:ミリ秒」の形式で表示されます。カスタム・バインディングはこの後の手順で実装します。

LazyInitEvalSolutionのすべての動作を実装するための手順は、以下のとおりです。

手順1：バインディングの遅延（lazy）評価と先行（eager）評価の違いの理解

バインディングの評価方法には、遅延評価と先行評価があります。それぞれ、以下のように動作します。

- バインディングの先行評価：バインディングが無効になるたびにバインディングの更新後の値が計算される
- バインディングの遅延評価：必要なときにのみバインディングの更新後の値が計算される

LazyInitEvalExerciseプロジェクトに含まれるStopWatchNode.javaファイルのコードが、このサンプルの初期コードになります。この演習では、実行する手順に合わせて、LazyInitEvalMain.javaのコードの抜粋を示していきます 。

バインディングの遅延評価の実装: StopWatchNode.javaのコードでは、リスト1に示すとおり、**SimpleStringProperty**のインスタンスを生成して**StopWatchModel**クラスの**elapsedMillisProperty**プロパティにバインドし、そのインスタンスに**InvalidationListener**を追加しています。

この場合、**elapsedTimeStrProperty**の値は、無効になってもすぐには評価されません。これが、「遅延評価」と呼ばれている理由です。

バインディングの先行評価の実装:リスト2に示すとおり、ユーザーが「**invalidate - click to bind/unbind**」テキストをクリックした場合、**elapsedNode**の**textProperty**が**elapsedTimeStrProp-**

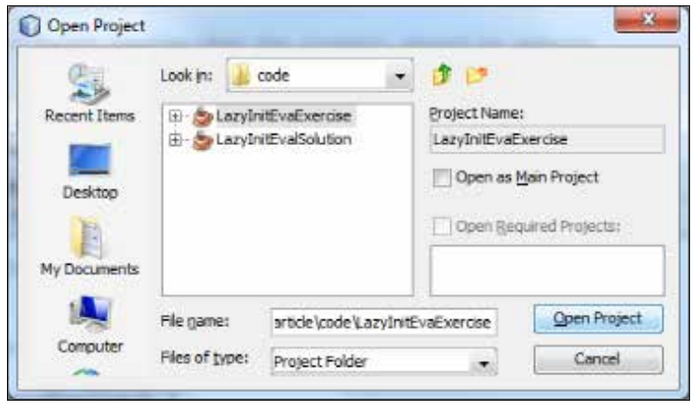

図4

図5

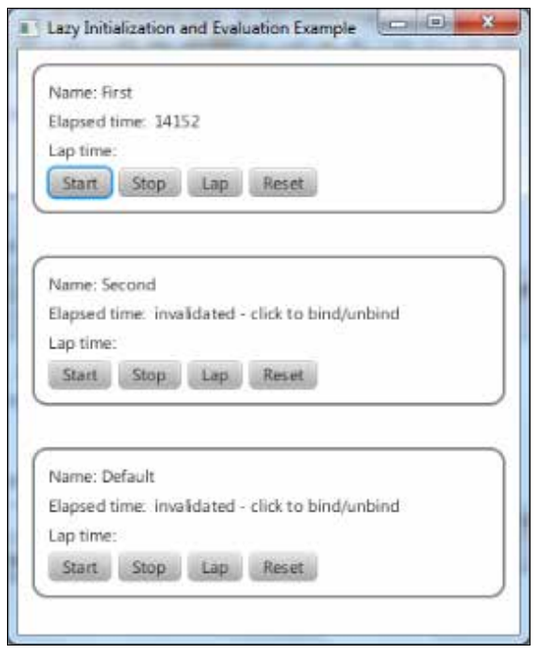

図6

ertyにバインドされます。この場合、バインディングは先行評価されます。つまり、**elapsedTimeStrProperty**の値が無効になるたびに、**elapsedNode**（**Label**コントロール）がその値を取得して表示します。

アプリケーションで遅延評価を適宜使用することにより、不要な処理を避けることができ、多くの場合パフォーマンスを向上できます。次に、もう1つの最適化技法である、プロパティの遅延初期化について説明します。

手順2:プロパティの遅延初期化の実装

JavaFXにおけるプロパティの初期化には、遅延初期化と先行初期化があります。遅延初期化にはいくつかの実装パターンがありますが、準遅延（half-lazy）と完全遅延（full-lazy）の2つが一般的です。

先行初期化の実装方法を**リスト3**に示します。LazyInitEvalExerciseプロジェクトのStopWatchModel.javaファイルの中で、3つのプロパティを定義しています。

それぞれのプロパティには、標準に従い、setterメソッド、getterメソッド、およびpropertyメソッドがあります。たとえば、リスト3の**lapMillis**プロパティには、**setLapMillis()**、**getLapMillis()**、**lapMillisProperty()**というメソッドがあります。これらのメソッドの実装では、**apMillis**プロパティが**setLapMillis()**、**getLapMillis()**、**lapMillisProperty()**のいずれかのメソッドによって最初にアクセスされたときに、**SimpleIntegerProperty**がインスタンス化されます。このような初期化方法では、プロパティの実際の使用法によって、メモリ・リソースが浪費される場合があります。この処理を最適化する方法を説明します。最初に準遅延初期化から見ていきます。

準遅延初期化の実装:リスト4に準遅延初期化の方法を示します。この場合、**getLapMillis()**を呼び出しても、**lapMillis**から参照される**SimpleIntegerProperty**はインスタンス化されません。また、**lapMillis**プロパティのデフォルト値を渡して**setLapMillis()**メソッドを呼び出した場合も、SimpleIntegerPropertyはインスタンス化されません。なお、lapMillisプロパティのデフォルト値はDEFAULT_LAP_MILLISに保持されています。

リスト4のコードをStopWatchModel.javaファイルに組み込んでください。次に、より積極的な遅延戦略について説明します。

完全遅延初期化の実装:リスト5に完全遅延初期化の方法を示します。この場合、**getName()**メソッドを呼び出しても、**name**から参照される**SimpleStringProperty**はインスタンス化されません。また、**setName()**メソッドを呼び出した場合でも、SimpleStringPropertyはインスタンス化されません。これには、**nameStr**という余分なインスタンス変数の導入というコストが伴いますが、それでも、通常はバインディングで使用されないプロパティに対してはこのような方法が適しています。

リスト5のコードをStopWatchModel.javaファイルに組み込んでください。次に、カスタム・バインディングの作成に関する概念の説明に移ります。

遅延は良いこと
アプリケーションで遅延評価を使用することにより、不要な処理を避けることができ、多くの場合**パフォーマンスを向上**できます。

手順3:カスタム・バインディングの作成

前述のとおり、経過時間の値は、**図6**では整数ですが、**図3**では「分:秒:ミリ秒」形式の文

リスト1 / リスト2 / リスト3 / リスト4 / リスト5

```
private StringProperty elapsedTimeStrProperty = new SimpleString-
Property();

... code omitted ...

    elapsedTimeStrProperty.addListener(new InvalidationListener() {
      public void invalidated(Observable observable) {
        if (!elapsedNode.textProperty().isBound()) {
          elapsedNode.setText("invalidated - click to bind/unbind");
        }
      }
    });

    elapsedTimeStrProperty.bind(stopWatchModel.elapsedMillisProp-
erty()
                        .asString());
```

全てのリストをテキストで表示

字列です。「分:秒:ミリ秒」形式を表示するためには、**SimpleIntegerProperty**のデフォルトのバインディングではなくカスタム・バインディングを使用します。

本サンプルでは、カスタム・バインディングを実装するために、**StringBinding**クラスを継承した**TimeStringBinding**クラスを定義しています。StringBindingクラスは、図1にあるBindingインタフェースを実装しています。**リスト6**に**TimeStringBinding** クラスのコードを示します。

リスト6に示すとおり、カスタム・バインディングを実装するためには、以下の作業を行います。

- バインディングのための入力引数を渡してスーパークラスの**bind()**メソッドを呼び出すコンストラクタを定義する
- バインディングで必要とされる出力値を返すように**computeValue()**メソッドをオーバーライドする

LazyInitEvalExerciseプロジェクトでこのカスタム・バインディングを使用するためには、まず、StopWatchNode.javaファイルの以下の行をコメントアウトします。

```
elapsedTimeStrProperty.bind(
  stopWatchModel.elapsedMillisProp
erty()
                          .asString()
);
```

次に、StopWatchNode.javaに以下の行を追加します。

```
elapsedTimeStrProperty.bind(
  new TimeStringBinding(
    stopWatchModel.elapsedMillis
Property()
  )
);
```

応用問題として、**図3**のとおりに表示されるように、**lapNode textProperty()**に対するバインディングも同様の方法で変更してください。

変更の完了後、LazyInitEvalExerciseアプリケーションを実行し、カスタム・バインディングによって経過時間とラップ・タイムの値が**図3**のような形式で表示されることを確認します。

まとめ

JavaFX 2には、プロパティとバインディングの強力なフレームワークを実現する、多数のクラスやインタフェースが含まれています。本記事で説明した遅延方式を利用することで、バインディングの評価とプロパティの初期化を最適化できます。さらに、JavaFX 2 APIのデフォルトのバインディング動作を拡張する必要がある場合は、カスタム・バインディングを定義できます。
</article>

LEARN MORE

- 「Using JavaFX Properties and Binding」
- Michael Heinrichsのブログ内のJavaFXのプロパティとバインディングに関する記事

リスト6

```
package javafxpert.lazyiniteval.binding;

import java.text.SimpleDateFormat;
import javafx.beans.binding.StringBinding;
import javafx.beans.property.IntegerProperty;

public class TimeStringBinding extends StringBinding {
  private SimpleDateFormat timeFormat = new
SimpleDateFormat("mm:ss:SSS");
  private IntegerProperty millis;

  public TimeStringBinding(IntegerProperty millisArg) {
    super.bind(millisArg);
    millis = millisArg;
  }

  @Override
  protected String computeValue() {
    int elapsedInt = millis.get();
    return timeFormat.format(elapsedInt);
  }
}
```

全てのリストをテキストで表示