

Java™ magazine

By the Java community 

The New Java Developers



1 JAVAMAILが届ける メッセージ

JavaMailとFaceletsを利用した電子メール送信Webアプリケーションの作成



5 アプリにWEBを取り 込もう

JavaFXのWebViewを使ってアプリにHTML、CSS、JavaScriptを埋め込めば、これまでにないアプリの開発が可能に



9 JAVAFXとSWINGの 統合

JavaFXにより、視覚効果を利用した動きのあるツールバーを実現



11 アスペクト指向プログラミングは必要か？

Java EEでは、明らかにAOPを使用すべき場面はほとんどない



16 GLASSFISHで実現する シンプルなクラスタリングと高可用性

GlassFishクラスタでアプリケーションの高可用性をサポートし、セッションのフェイルオーバーを実現





T. LAMINE BA

BIO

JavaMailが届けるメッセージ

JavaMailとFaceletsを利用した電子メール送信Webアプリケーションの作成

この記事では、コアJavaMail APIを利用して電子メールを送信する、シンプルなWebアプリケーションを作成します。このアプリケーションには、3つのWebページが含まれます。それぞれトップページ、「送信完了」の確認ページ、「送信失敗」の通知ページです。

注:アプリケーションのソース・コードは、こちらからダウンロードできます。Webアプリケーションのトップページ(図1参照)には、次のWebコンポーネントが含まれます。

- 送信者の電子メール・アドレスの入力フィールド
- 受信者の電子メール・アドレスの入力フィールド
- 電子メールの件名の入力フィールド
- 電子メールの本文の入力フィールド
- Simple Mail Transfer Protocol (SMTP)サーバーのIPアドレスまたはホスト名の入力フィールド
- SMTPサーバーが要求するユーザー名の入力フィールド
- SMTPサーバーが要求するパスワードの入力フィールド

テキストだけではありません
JavaMail APIは、JavaBeans Activation Frameworkを利用して、プレーンなテキスト以外のコンテンツも扱えます

- SMTPサーバーが使用するポート番号の入力フィールド
 - 電子メール送信ボタン
- 確認ページ(図2参照)と通知ページ(図3参照)では、ユーザーがトップページに戻るためのボタンのみ必要です。

JavaMail API

JavaMail APIは、電子メッセージの読み取り、作成、送信などの一般的な電子メール機能を提供するパッケージです。JavaMailはプラットフォーム非依存、プロトコル非依存のフレームワークであり、Java EEに含まれています。

図4に示すとおり、JavaMailには、アプリケーション・コンポーネントが電子メールの送受信に使用するアプリケーション・レベルのインタフェースがあります。また、プロトコル固有の処理に対応するサービス・プロバイダ(SP)インタフェースもあります。たとえば、SMTPは電子メールの送信に使用されるプロトコルです。Post Office Protocol 3(POP3)は、電子メール受信用の標準になっています。Internet

Mail Access Protocol (IMAP)は、POP3の代わりに使用できます。

さらに、JavaMail APIにはJavaBeans Activation Framework (JAF)が含まれており、Multipurpose Internet Mail Extension (MIME)、URL、添付ファイルなど、プレーンなテキスト以外の電子メール・コンテンツを扱えます。

必要なソフトウェア

このチュートリアルでは、次のソフトウェアを使用します。

- Microsoft Windowsプラットフォーム(ここではMicrosoft Windows 7を使用)
- Java EE 6以降のJDK
- NetBeans IDEなどの一般的なIDE(ここではNetBeans IDEバージョン7を使用)
- Webサーバー(Oracle GlassFish Server、GlassFish Server Open Source Edition、Apache Tomcatなど)

方法

このチュートリアルではJavaServer

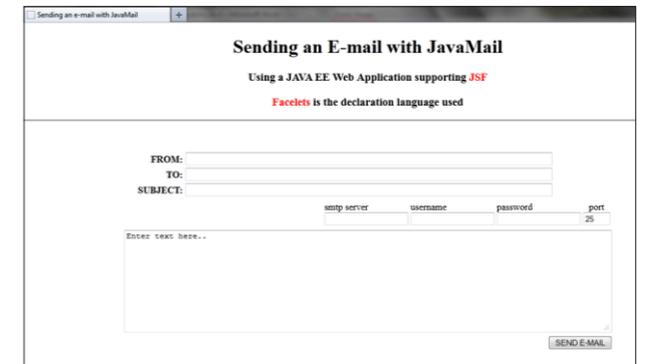


図1

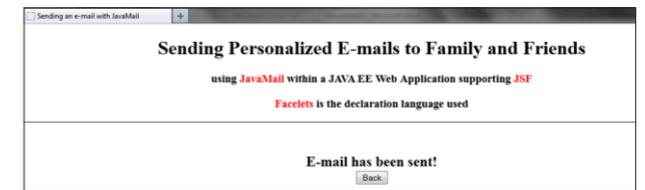


図2



図3

Faces(JSF)テクノロジーを利用してWebアプリケーションを作成します。そのため、次のワークフローを進めることにします。

- 手順1: バッキングBeanを作成する
- 手順2: コンポーネント・タグを使用してWebページを作成する





JOSH MARINACCI



アプリにWebを取り込もう

JavaFXのWebViewを使ってアプリにHTML、CSS、JavaScriptを埋め込めば、これまでにないアプリの開発が可能に

JavaFX 2.0は、リッチ・クライアント・アプリケーションの開発方法を「再発明」します。新しい機能とグラフィック基盤により、開発者は魅力的で反応のよい、優れたインターフェースを作成できるようになります。JavaFX 2.0の優れた新機能の1つに、**WebView**コントロールがあります。**WebView**を使用すると、アプリケーションの中にHTMLコンテンツをそのまま埋め込むことができるため、デスクトップとWebが協調するまったく新たな世界が開かれます。

JavaFXチームは、開発者がHTMLを、本来のWebブラウザでの機能と正確さをそのままに、直接アプリケーションに埋め込むことができるようにしたいと考えました。これは、従来のSwing **JEditorPane**ではできなかったことです。そのために、オープンソースのHTMLレンダリング・エンジンである**WebKit**を利用することにしましたこれは、さまざまなデスクトップおよびモバイル・プラットフォームで、急速に標準的なレンダラとなっています。Safari、Google Chrome、iOS、Android、webOS、さらにはAmazon Kindleでも使われています。

JavaFXチームはまた、新しいWebコントロールを、ネイティブ**WebKit**ラ

イブラリの単なるラッパー以上のものにしたいと考えました。**WebView**は、画面上の描画とすべてのネットワーク・アクセスにJavaを使用することで、単なるラッパーで可能なことに比べ、ずっと緊密な統合を実現しています。**WebView**では、この回転するWebページの例が示すように、3Dシーンの中に描画することもできます。

WebViewコントロールのさまざまな使用方法

WebViewをアプリケーションで使用するのとは、とても簡単です。**WebView** Javaオブジェクトを作成し、そこから実際に**WebKit**に接続するJavaオブジェクトである**WebEngine**への参照を取得するだけです。ページをロードするには、WebページのURLを用いて**engine.load()**を呼び出します。**engine.loadContent()**を使用して、アプリ内で文字列から生成したコンテンツを直接ロードすることもできます。そして、作成した**WebView**を、他のコントロールと同じようにJavaFXシーンに配置します。これで完了です。

もちろん、**WebView**のAPIがそれだけのものではあれば、この記事は必要ないでしょう。Webコンテンツをア

プリの中に持ち込むことができたら、それで実際に何ができるのでしょうか。おもなユースケースには、次の4つがあると思います。

- Webの一部をデスクトップ・アプリに埋め込む。Google マップをアプリに埋め込むのが良い例です。
- Javaを使用してリモートのWebコンテンツを操作する。つまり、コンテンツはWebからコンテンツをロードしますが、Javaコードを使ってそれを変更します。
- 内部で生成したコンテンツやローカルにロードしたコンテンツを表示する。リアルタイムのレポート生成が良い例です。
- JavaScript側からJavaコンテンツを操作する。**WebView**では、JavaScriptからJava側にアクセスする方法は提供されていませんが、少し頭をひねればいろいろと面白いことができます。

Webコンテンツの読取りと埋込み

最初のユースケースから始めましょう。Webの一部をデスクトップ・アプリに埋め込みます。単にWebページを

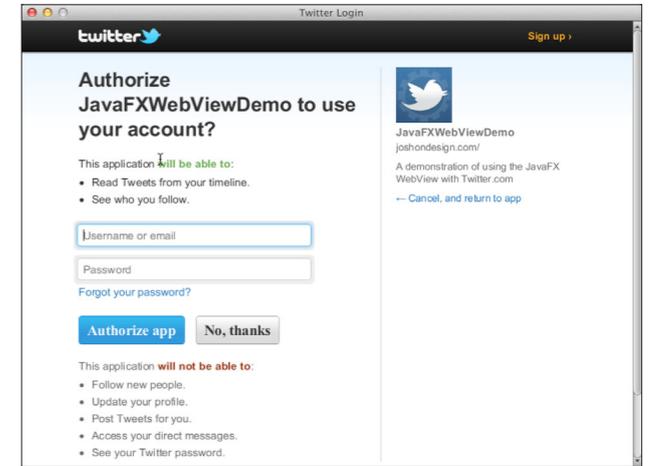


図1

ラップするのではなく、私が本当に悩まされている問題であるTwitterの認証に対処できたら、素晴らしいことではないでしょうか。

Twitterは、OAuthを使用してサード・パーティのアプリを認証していません。OAuthでは、アプリケーションはユーザーをTwitter.com上の特別なURLへ転送する必要があります。そのページでユーザーがアプリを承認すると(図1を参照)、ユーザーはアクセスを要求したアプリに戻ります。これは、アプリがWebサイトであれば問題ありませんが、アプリがデスクトップ・プログラムである場合、ユーザーが戻るべきサーバーURLがありません。この問題に対処するため、Twitter

写真:CHRIS PIETSCH / GETTY IMAGES



けです。そこで、最初にすべてのコードを **StringBuffer** に入れます(これをより強力なバージョンにするとしたら、JavaScriptをコードの中で生成するのではなく、ファイルからロードすることができます)。スクリプトの準備ができれば、**engine.executeScript()** を呼び出すことで注入できます。このメソッドは、コードの戻り値をラップしているオブジェクトを返します。この例では、**body**s オブジェクトです。この戻り値を使用して、JavaScript

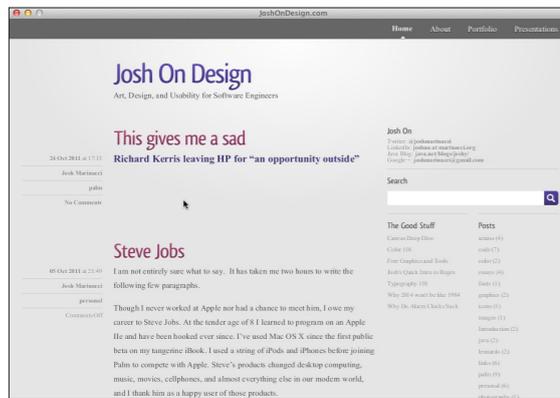


図3

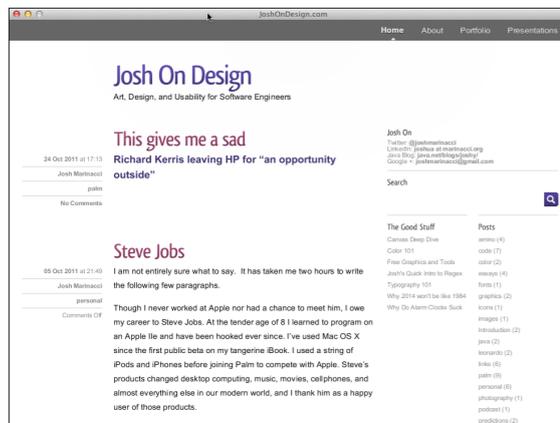


図4

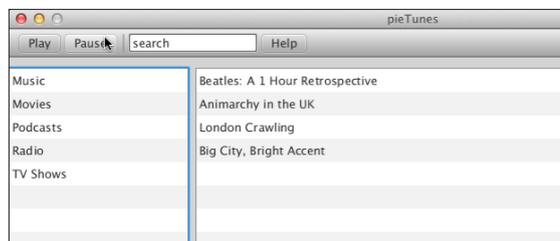


図5

側からJava側に簡単なデータを渡すことができますが、この例では、コードが正常に実行されたことを確かめているだけです。

図3に、私のブログのデザインを示します。そして図4に、新しいスタイルのブログを示します。

生成されたコンテンツの操作

最後の例として、**WebView** を使ってアプリ自身が持つコンテンツを表示するヘルプ・システム

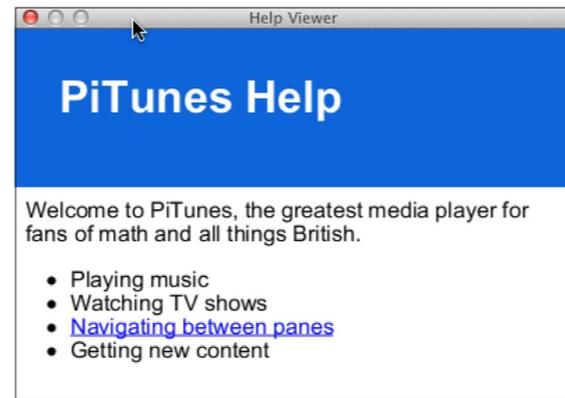


図6

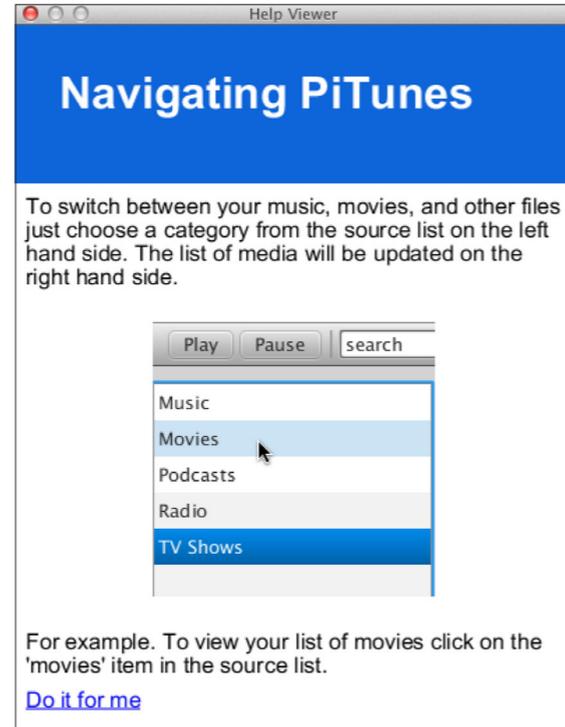


図7

リスト7 リスト8

```
private void modifyDoc(Document newDoc, WebEngine engine) {
    StringBuffer script = new StringBuffer();
    //grab the body
    script.append("var bodys = document.getElementsByTagName('body');");
    //change the colors
    script.append("bodys[0].style.backgroundColor = '#FFFFFF';");
    script.append("bodys[0].style.color = '#000000';");
    //set a new default font
    script.append("bodys[0].style.fontFamily = 'sans-serif';");

    script.append("bodys;");
    //execute
    Object retval = engine.executeScript(script.toString());
    //return value doesn't matter in this case
    p("return value = " + retval);
    JSONObject obj = (JSONObject) retval;
}
```

全てのリストをテキストで表示

ムを示します。かつては、アプリの中でリッチテキストのヘルプを表示するには、独自のヘルプ・エンジンを使用する必要がありました。しかし、今ではすべてをHTMLとして記述するだけで、それをウィンドウに表示できます。

この例では、PiTunesという架空のメディア・プレーヤを作成しました。図5に示すように、最上部にツールバーがあり、左にメディア・タイプのリスト(ソース・リストと呼びます)、右にメディア・ファイルのリストがあります。

ムービーのリストを表示するには、リストにある「Movies」という項目をクリックします。しかし、この操作は分かりにくいかもしれないので、説明するためのオンライン・ヘルプを追加します。ユーザーが「Help」ボタンをクリックすると、ヘルプ・コンテンツを表示する第2のウィンドウが開きます。リスト8を参照してください。

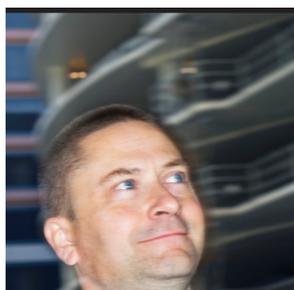
ここで注意すべき重要なことが2つあります。第1に、ヘルプ・ファイルを、

getResource() で得られるURLからロードしているという点です。これは、ディスク上のコンテンツだけでなく、JARファイル内部のコンテンツもロードできることを意味しています。第2に、最初のページでリンクをクリックして次のページに移ると、画像が表示されるという点です。この画像は、リソース・パッケージの中にあるPNGファイルです。マークアップでは、**** のように、これをローカルに参照しています。

WebView は、すべてのリソースのロードにJavaを使用するため、JARファイルやデータベースの中などJavaに特有の場所やデータベースを含め、あらゆるところからコンテンツをロードできます。すべてJavaと同様です。図6と図7を参照してください。

これで、リッチなヘルプ・コンテンツをアプリの中に入れることができました。さらに一段階進めることもできます。何かを実行する方法をユーザーに伝えるだけでなく、それを





SIMON RITTER



パート3 JavaFXとSwingの統合

JavaFXにより、視覚効果を利用した動きのあるツールバーを実現

この記事は、JavaFXと既存のSwingアプリケーションとの統合に関する全3回シリーズの最終回です。パート1では、JavaFXシーンをJComponentとしてラップするための基本事項とイベントの処理方法を見ました。パート2では、サンプルアプリケーションの中でJavaFXのテーブルを使用する方法を見ました。今回、株式の詳細情報を提供する同じサンプルアプリケーションのための面白いツールバーを作成するために、視覚効果を利用します。

注: サンプルアプリケーションのコードは、こちらからダウンロードできます。

このアプリケーションには、ポートフォリオの表示、株価の更新、新規ポートフォリオの作成といったコマン

ドにすばやくアクセスするためのアイコンを用いたツールバーがあります。この機能は、SwingのJToolBarコンポーネントを利用しています。このツールバーを視覚的により楽しくするため、すべての機能を維持しながら、JavaFXのToolBarコントロールに置き換えます。

機能の追加

まずJavaFXのコードから始め、その後でコードをアプリケーションに統合する方法を説明します。アイコンに機能を追加するため、ActiveIconという新しいクラスを作成します。コンストラクタの概要をリスト1に示します。

リスト1のコードは単純に、コンストラクタに渡される画像名を引数と

してとり、ツールバーで使用できるImageViewノードを作成しています。また、これから作成するエフェクトの1つのために、画像の幅も必要です。コードから分かるように、アイコンに設定可能なエフェクトとして、フェード、スピン、グロー、スライドの4種類を定義しています。

これらのエフェクトの実装をリスト2とリスト3に示します。これもコンストラクタの一部です。

視覚効果のためには、Timelineを利用してアイコンのプロパティを変更します。使用するプロパティはそれぞれがDoublePropertyであり、ImageViewオブジェクトの適切なメソッドを呼び出すことでアクセスできます。プロパティを取得したら、0秒に1つのKeyFrameがあり、1,000ミリ秒(1秒)にもう1つのKeyFrameがあるTimelineを作成します。それぞれのKeyFrameでKeyValueを使用して、変更の対象とするプロパティを関連付け、Timelineで定義した時点におけるそのプロパティの値を定義します。エフェクトを連続的にするため、Timelineに自動反転を設定し、繰り返し回数を無限に設定します。タイムラインの中で変更されるアイコンのプロパティとその値は、暗黙的にバインドしています。

フェード・エフェクトではアイコンの

透明度が変化し、グロー・エフェクトではx軸とy軸の拡大率が変わり、スピン・エフェクトではアイコンが360度回転します。

スライド・エフェクトはもう少し複雑で、アイコンがy軸に沿ってアイコンの幅の2倍だけ移動します。他のアイコンと重ならないように、アイコンの背面に透明な四角形を配置して、十分なスペースを確保します。そのために、この四角形の不透明度を0に設定します(visibilityプロパティをfalseに設定するのではうまくいかないことに注意してください)。

新しいツールバーの作成

これで、リスト4に示すように、ActiveIconクラスを使用してツールバーを作成できます。リスト4のコードでは、StocksMonitorMainWindowクラスに、メインGUIの作成を担当する新しいメソッドを追加しています。ここでは、パート1で説明した手法を使用して、JavaFXノードをラップし、他のComponentオブジェクトと同様にSwing GUIで操作できるJFXPanelを作成します。

ツールバー・ノードを作成するには、JFXPanelの新規finalインスタンスを作成し、JavaFXのスレッドで実行するタスクを設定します。JavaFX



JavaFXとSwingに関する全3回の連載記事のハイライトを語るSimon Ritter

写真: BOB ADLER





ADAM BIEN



アスペクト指向プログラミングは必要か？

Java EEでは、明らかにAOPを使用すべき場面はほとんどない

アスペクト指向プログラミングの目的は、再利用可能な機能をビジネス・ロジックから分離することです。Wikipediaには、次のように書かれています。「コンピューティング分野におけるアスペクト指向プログラミング(AOP)は、クラス間を横断するような機能を分離することでモジュール性を高めることを目的としたプログラミング・パラダイムである。AOPはアスペクト指向ソフトウェア開発の基礎となっている」

AOPは数年前に話題となったものの、J2EEで盛んに使用されるようにはなりません。それには理由があります。

注:この記事のサンプルJava EEプロジェクト([AOPandJavaEE](#))はProject KenaiのWebページから入手できます。

アスペクトはコモディティ

アスペクトは、1998年初めのEnterprise JavaBeans(EJB)1.0仕様

(JSR 318)の不可欠な部分でした。開発者は、BeanクラスとRemoteおよびHomeインタフェースを開発する必要がありましたが、横断的なアスペクトの開発は不要でした。スレッド管理、プーリング、セキュリティ、トランザクション、セッション処理、そして永続化までもが、ビジネス・ロジックから明確に分離された構成可能なアスペクトでした。初期のEJB 1.0の時代は、横断的な機能をPOJOで構成できました。

EJB 1.1とJ2EEが登場すると、JavaクラスはXMLに置き換わり、最近のJava EE 5およびJava EE 6仕様ではXMLが必須ではなくなりました。大部分のユースケースをアノテーションで構成でき、さらにXMLを使用してアノテーションをオーバーライドできます。

EJB 1.0の全体的な考え方はアスペクトに基づいていますが、アスペクトという語自体は仕様に2回しか登場せず、その文脈も異なります。AOPと

普及には至らず AOPは数年前に話題となったものの、J2EEで盛んに使用されるようにはなりません。それには十分な理由があります。

EJB 1.0はほぼ同時期に「発明」されたのです。

J2EEでは、EJB Beanをトランザクションなどによって構成可能な方法でデコレートすることが、開発における不可欠な部分でした。他のフレームワークも同じ機能を提供していましたが、その機能をAOPと呼んでいました。

EJB 3.1では、単純なBeanを定義することで、さまざまなアスペクトを利用できます。たとえば、EJB 3.1のステートレスBeanには、最初からトランザクションとライフサイクルのアスペクトが備わっています。

```
@Stateless public class Greeter {
    public String greet() {
        return "Morning!";
    }
}
```

このgreet()メソッドは、直接呼び出されることはありません。コンテナがプールからEJBインスタンスを取り出し、呼び出しをディスパッチします。また、各メソッド呼び出しの前にトランザクションを開始します。メソッドの

呼び出しは、デフォルトの構成可能なアスペクトのチェーンによって処理され、最終的に実際のビジネス・ロジックが実行されます。アプリケーション・サーバーの外側で同じ機能を実装しようとするならば、デコレータ・パターンや、AOPなどの汎用的なソリューションで実装することになるでしょう。

汎用デコレータ:インターセプタ

リスト1に示すように、このGreeter EJB Beanを直接サブレットにインジェクションした場合でも、呼び出しをデコレートできます。

インターセプタは、JDK 1.3で導入されたProxyクラスに似ていますが、より使いやすくなっています。必要なことは、1つのメソッドを持つ1つのクラスを実装することだけです。リスト2に、一般的なビジネス・メソッド・インターセプタを示します。

メソッド名は自由に決められますが、戻り型、例外、パラメータはリスト2に定義されているシグネチャと同じにする必要があります。インターセプタは、@InterceptorsアノテーションがマネージドBeanまたはEJB Beanと関連付けられます。



