FaceletsはぴったりとJSFにフィットします

JSF専用に作られたビュー技術がついに登場

Richard Hightower (rhightower@arc-mind.com)
Developer
ArcMind Inc.

2006年 2月 21日

JSFとJSPを組み合わせることは、靴べらを使って手袋の中に足を突っ込もうとするようなものです。つまり可能ではありますが、より良いものが現れるまでの、その場しのぎにすぎません。この記事では、熱狂的なJSF支持者であるRick Hightowerが、Faceletsの素晴らしさについて、つまり容易なHTML風テンプレートと再利用可能な構成コンポーネントについて解説します。

私は最近、あるJSF（Java™Server Faces）プロジェクトに関わる中で、初めてFaceletsを使う楽しみを経験しました。私がFaceletsに関して最も素晴らしいと思うのは、再利用可能な構成コンポーネント（composition component）が作れることです。ページ（JSPなど）からコンポーネントを作れる、ということによって、私のJSF開発は大いに楽になりました。そして私の得た結論は、もし皆さんがまだFaceletsを使っていないのであれば、皆さんはJSFから最大限のものを引き出しきっていないのです。

JSFとJavaServer Pages技術のミスマッチは、JSF開発では深刻な問題です。つまりJSPによる動的コンテンツを、JSFによるコンポーネント・ベース・モデルの中にいかに統合するか、という問題です。JSPはひたすら動的出力を生成することに集中する一方、JSFはJSPがコンポーネント・モデル構築を調整するように要求します。こうした食い違いが起きる原因は、そうした課題が元々のJSPでは想定されていなかったためです。

ほとんどのJSF開発者は、そうした問題に対して、その場限りの方法で対処しています。しかしそうした方法は、金槌で頭を叩いても怪我をしないように、羽根枕で金槌を包むのと似ています。Faceletsは、そんな方法よりも、ずっと体系的なソリューションです。つまりFaceletsは、JSFコンポーネント・モデルに適したテンプレート言語なのです。

Faceletsは、次のように幾つかの魅力的な機能を備えています。

- （タイルのような）テンプレート
- 構成コンポーネント
- カスタム・ロジック・タグ

- 表現機能
- 設計者に使いやすいページ開発
- コンポーネント・ライブラリー作成

こうした機能は、皆さんが想像するよりもはるかに関連しあっており、そして統合されています。この記事では、最初の2つ、つまりテンプレート動作と構成コンポーネントを取り上げます。私が以前、『JSFを信じない人のために』というシリーズ記事で開発したサンプルを元にしたWebアプリケーションを使って、タイルではなくFaceletsビューを使うように変更します。この先に進む前に、サンプルコードをダウンロードしてください。また、解説を理解するためには、Faceletsもインストールする必要があります。

Faceletsの概要

皆さんは、Faceletsはタイルの置き換えでしかない、と思ってしまうかも知れませんが、それは大きな間違いです。Faceletsはタイルをはるかに上回り、JSFに対する全く新しい考え方なのです。

マークアップを選ぶ

ほとんどの開発者はFaceletsでXHTMLを使いますが、実際にはFaceletsのフレームワークはマークアップには依存しません。FaceletsはXUL（XULFaces）と互換性があり、Kito Mannは、これを使ってJSF Central用のRSSフィードを作っています。

JSPは、サーブレットを生成するテンプレート言語です。JSPのボディーは、サーブレットのdoGet() メソッドやdoPost() メソッドと等価なものになります（つまりjspService() メソッドになります）。JSFカスタム・タグ（f:viewやh:formなど）は、現在の状態での自分たちを描画するための、単なるJSFコンポーネントへのコールです。JSFコンポーネント・モデルのライフサイクルは、JSPに生成されたサーブレットのライフサイクルとは独立しています。この独立性のために、誤解が生じやすいのです。

FaceletsはJSPとは異なり、JSFコンポーネントのライフサイクルを念頭に、ゼロから構築されたテンプレート言語です。Faceletsを使用して、（サーブレットではなく）コンポーネント・ツリーを構築するテンプレートを作成するのです。つまり他のコンポーネントを組み合わせることでコンポーネントを構成できるため、再利用がずっと容易になるのです。

Faceletsを使うと、JSFコンポーネントを使うためのカスタム・タグを書く必要がありません。Faceletsは、JSFカスタム・コンポーネントをネイティブで使用します。JSFとFaceletsとの橋渡しのためには、特別なコーディングはほとんど必要ありません。必要なことは、Faceletsタグ・ライブラリー・ファイルの中でJSFコンポーネントを宣言することだけです。何ら追加の開発を行わなくても、Faceletsテンプレート言語の中では直接JSFコンポーネントを使えるのです。

Faceletsテンプレートのフレームワーク

Faceletsは、コンポーネント構築のためのテンプレート・フレームワークを提供しているという点で、Tapestry（参考文献）と似ています。しかし私達のようにJSPを背景に持つ者にとっては、FaceletsはTapesryよりもずっと使いやすく思えます。Faceletsを使えば、皆さんには既におなじみのJSTL風のタグや、JSTL/JSF/JSP風の表現言語を処理することができます。それに、簡単に理解できるということは、すぐにFaceletsを使った開発が始められるということです。

FaceletsとTapestry

FaceletsとTapestryは比較したくなるほど似ています。実際Tapestryは、登場した時点では時代を先取りしており、FaceletsもTapestryの考え方を一部採用しています。しかし、Faceletsを単純にTapestryのJSF版と考えてしまうのは間違いです。2つの技術は別物なのです。Tapestryについて学ぶには、Brett McLaughlinによる2回シリーズの記事、「In tune with Tapestry」を読んでください。

Faceletsを使うと、ページの中に直接含められる、あるいはFaceletsタグ・ライブラリーに容易に追加できるコンポーネント・アセンブリーを定義することができます。実際のところFaceletsでは、驚くほど素早くカスタム・タグ（JSPカスタム・タグに似た構成コンポーネントやタグ）を定義することができます。Faceletsでは、こうしたコンポーネント・アセンブリーを使ってサイト・テンプレート（そして、それよりも小さなテンプレート）を定義することができます。これはタイルを使って作業する場合と似ていますが、定義ファイルはありません。またFacelets APIでは統合が容易なインターフェースが用意されているため、カスタムJSFコンポーネントの中でFaceletsを使うこともできます。

タイルからFaceletsへ

先に触れた通り、ここで使用するWebアプリケーション例は、私が『JSFを信じない人のために』シリーズの中で作ったものです。これは、オンラインCD店の在庫を管理するための、CDリストに対するCRUD（create, read, update, and delete）アプリケーションです。これにはユーザーが新しいCDをシステムに入力するためのフォームと、音楽カテゴリーを選択するための一連のラジオ・ボタンが含まれています。ユーザーがカテゴリーを選択すると、あるJavaScriptが起動され、そのフォームが即座にサーバーにポストされます。このアプリケーションにはCDリストも含まれており、ユーザーはそのリストを使って、タイトルまたは演奏者でCDをソートすることができます。図1は、このアプリケーション・クラスのUML図です。

図1. オンラインCD店のクラス図

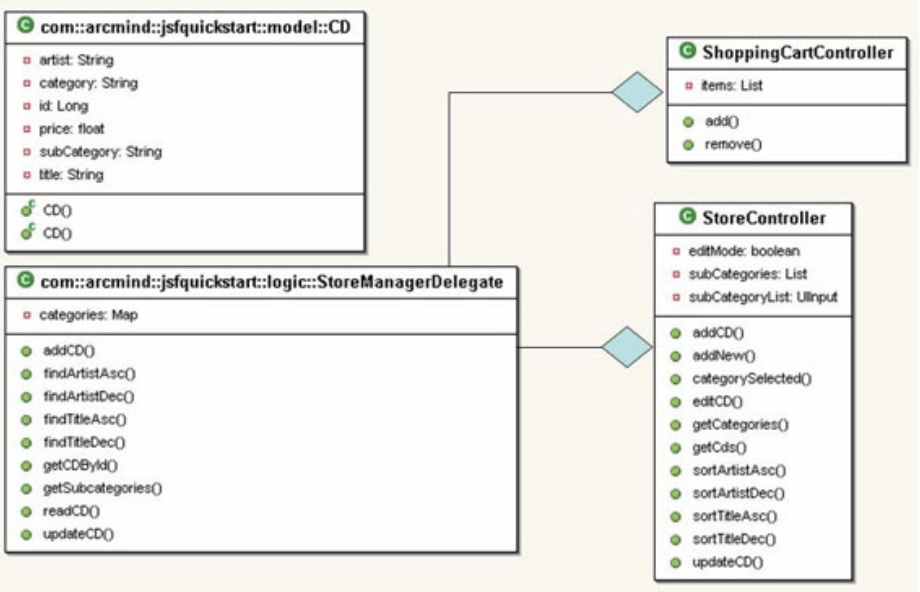
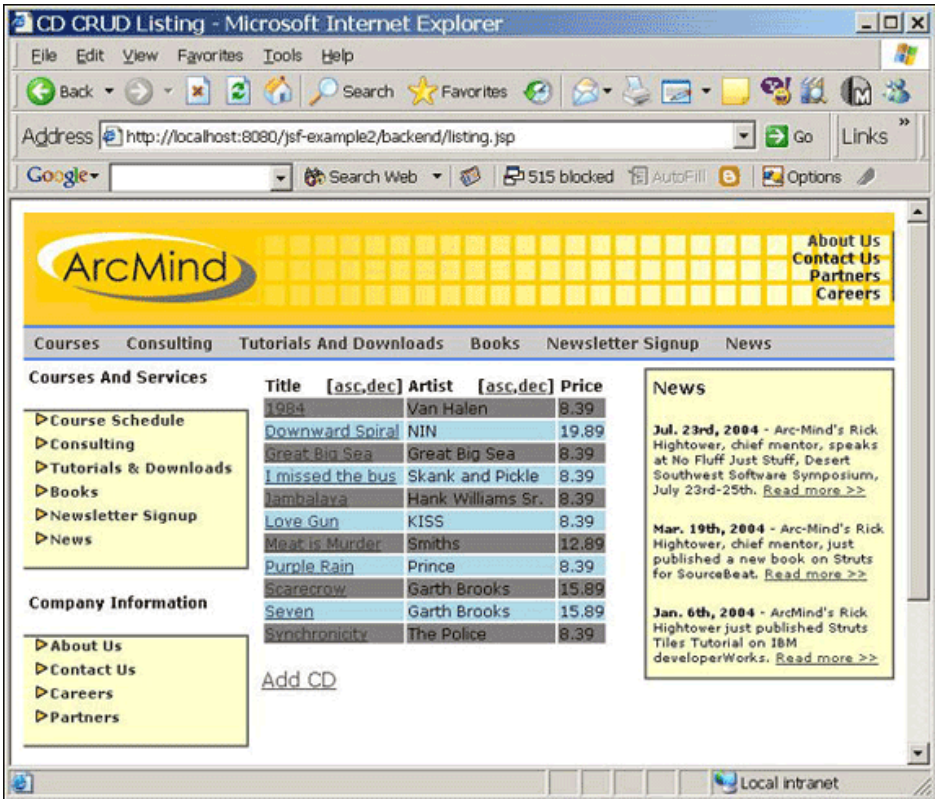


図2は、この店のCDリストのページを見たものです。

図2. このCD店のリスト・ページ



オリジナルのアプリケーションでは、タイルのビュー・サポート機能を使っていますが、ここではFaceletsを使ってビューを構築します。最初に、以前の例でのタイル・サポートをFaceletsで置き換え、次に構成コンポーネントを書くことにします。しかし何よりも前に、Faceletsをインストールしておく必要があります。

Faceletsをインストールする

Faceletsをインストールするための手順は簡単です。ここでは、サンプル・アプリケーションは既にダウンロードされ、インストールされている、という前提で説明しますので注意してください。

1. Faceletsディストリビューションをダウンロードし、解凍します。
2. jsf-facelets.jarをWEB-INF/libディレクトリーの中にコピーします（アプリケーションがデプロイされると、WEB-INF/libディレクトリーの中に入るはずです）。
3. Facelet initパラメーターをweb.xmlファイルに追加します。
4. FaceletViewHandlerをfaces-config.xmlファイルに追加します。

ステップ1と2は基本的なことなので、残り2つのステップの詳細について説明しましょう。

initパラメーターを追加する

このステップでは、実働のJSFアプリケーション（オンラインCD店の例など）が既にインストールされていること、そして（下記のパラメーターを追加することで）既存のweb.xmlページを編集中であることを想定しています。

```
<context-param>
 <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
 <param-value>.xhtml</param-value>
</context-param>
```

これはJSFに対して、Faceletのレンダラーが解釈できる『xhtml』という接頭辞を想定するように伝えています。

Faceletsには多くのパラメーターがあります。その完全なリストは、参考文献を見てください。このサンプルで何か問題がある場合には、DEVELOPMENT initパラメーターを参照するとデバッグに役立つでしょう。また、開発中にはREFRESH_PERIODパラメーターをlowに設定すると良いでしょう。

FaceletViewHandlerを追加する

```
  <application>
    <locale-config>
      <default-locale>en</default-locale>
    </locale-config>
 <view-handler>com.sun.facelets.FaceletViewHandler</view-handler>
  </application>
```

Faceletsテンプレートを有効にするためには、JSFに対してFaceletsビュー・ハンドラーのことを伝える必要があります。JSFのViewHandlerは、（Faceletsを含めた）様々なレスポンス生成技術に対するJSFリクエスト処理ライフサイクルの、Render ResponseフェーズとRestore Viewフェーズを処理するプラグインです。（JSFは拡張できないと思っている人は、誤った情報を信じているのです！）次のようなビュー・ハンドラーをfaces-config.xmlに追加することによって、FaceletsをJSFにプラグインします。

Faceletsでのテンプレート

Faceletsテンプレート・フレームワークは比較的理解しやすいので、まずこれについて説明しましょう。Faceletsテンプレートを作成し、使用するための手順は次の通りです。

1. layout.xhtmlページを作成します。
2. Faceletsの名前空間を定義することによって、Faceletsの使い方をインポートします。
3. ui:insertタグを使って、そのページの論理領域を定義します。
4. プレーン・テキストとui:includeタグを使って、適当なデフォルトを定義します。

オンラインCD店のListingページをレイアウト例に使って、この手順を1つずつ説明しましょう。

ステップ1.　layout.xhtmlページを作成する

layout.xhtmlページは、単なる通常のXHTMLテキスト・ファイルであり、次のようなdoctype宣言を使用しています。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
</html>
```

これ以上の説明は不要でしょう。

ステップ2. Faceletsの名前空間を定義する

テンプレート用にFaceletsタグを使うためには、下記のようにXML名前空間を使ってインポートする必要があります。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
...
```

ui名前空間の定義に注意してください。

ステップ3. ui:insertタグを使ってページの論理領域を定義する

次に、レイアウトの論理領域（タイトルやヘッダー、ナビゲーション、内容など）を定義します。下記はタイトル定義の方法を示す一例です。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
  <title><ui:insert name="title">Default title</ui:insert></title>
  <link rel="stylesheet" type="text/css" href="./css/main.css"/>
</head>
...
```

タイトルの論理領域を定義するためのui:insertタグの使い方に注意してください。ui:insert要素の内側の「Default title」という文字列は、このテンプレートを使う人がタイトルを渡さない場合の文字列を定義しています。上記は、次のように書くこともできます。

```
<title>#{title}</title>
```

ステップ4.プレーン・テキストとui:includeタグを使ってデフォルトを定義する

デフォルトとして、プレーン・テキスト以上のものを渡すこともできます。例えば、layout.xhtmlからの下記のコード断片を考えてみてください。

```
<div id="header">
    <ui:insert name="header">
     <ui:include src="header.xhtml"/>
    </ui:insert>
</div>
```

ここで私は、ui:insertタグを使って論理領域を定義し、ui:includeタグを使ってデフォルトを挿入しています。レイアウトを使用するページは、デフォルトとしてheader.xhtmlの内容をヘッダー・テキストとして使っていますが、ヘッダーはui:insertが定義する論理領域なので、このテンプレートを使うページは別のヘッダーを渡すこともできます。フロント・エンド（買い物カゴ付きのカタログなど）を持つアプリケーションや、（新製品追加用の）バックエンド管理を持つアプリケーションでは、バックエンド・サイトはヘッダーやナビゲーションの中に別々なリンクを持つこと

ができます。ui:includeタグを使うと、デフォルト・ヘッダーを新しいヘッダーに容易に交換できるのです。

リスト1は、サンプル・アプリケーションのListingページ、list. xhtmlのコード全体を示しています。

リスト1. list.xhtmlの全体

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
  <title><ui:insert name="title">Default title</ui:insert></title>
  <link rel="stylesheet" type="text/css" href="./css/main.css"/>
</head>

<body>

<div id="header">
    <ui:insert name="header">
     <ui:include src="header.xhtml"/>
    </ui:insert>
</div>


<div id="left">
  <ui:insert name="navigation" >
    <ui:include src="navigation.xhtml"/>
  </ui:insert>
</div>


<div id="center">
  <br />
  <span class="titleText"> <ui:insert name="title" /> </span>
  <hr />
  <ui:insert name="content">
   <div>
    <ui:include src="content.xhtml"/>
    </div>
  </ui:insert>
</div>

<div id="right">
  <ui:insert name="news">
    <ui:include src="news.xhtml"/>
  </ui:insert>
</div>

<div id="footer">
  <ui:insert name="footer">
    <ui:include src="footer.xhtml"/>
  </ui:insert>
</div>

</body>

</html>
```

これでレイアウトの定義方法は分かったので、今度はその使い方を説明しましょう。

Faceletsテンプレートを使う

テンプレートを呼び出すためには、ui:compositionタグを使います。テンプレートに引数を渡すには、ui:compositionタグのサブ要素であるui:defineタグを使います。リスト2では、オンラインCD店のサンプルでのレイアウト・ページを呼んでいます。

リスト2. レイアウト・ページを呼ぶ

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">


<ui:composition template="/WEB-INF/layout/layout.xhtml">
  <ui:define name="title">CD form</ui:define>
  <ui:define name="content">

 <!-- use the form tag to set up this form -->
 <h:form id="cdForm">

  ...
  ...
...

 </h:form>
  </ui:define>
</ui:composition>
</html>
```

上記の呼び出しには下記の名前空間が含まれていることに注意してください。

- xmlns:h="http://java.sun.com/jsf/html"
- xmlns:f="http://java.sun.com/jsf/core"

FaceletsではJSFタグ・ライブラリーに頼りません。そのためコア・コンポーネントやHTML JSFコンポーネントを使うためには、上記の名前空間を使ってインポートする必要があります。

htmlタグを使うことは変に思えるかも知れません。つまりリスト2に示すレイアウト・ページは、既にhtmlタグを持っているテンプレートを呼び出しているので、これは2つのhtmlタグを取得することを意味するのでしょうか。実は、ui:compositionタグの外側にあるものは全て無視されるのです。ですからhtmlタグがすることというのは、HTMLエディターからHTML断片が見えるようにすることだけなのです。これはランタイムの振る舞いには影響を与えません。

位置こそがすべて

ページがレイアウト・テンプレートを呼び出す場合には、そのページは単にテンプレートの位置を規定しさえすればよいのです（下記）。

```
<ui:composition template="/WEB-INF/layout/layout.xhtml">
```

このタグは、リスト1に示すテンプレートを呼び出します。ですから必要なこととしては、テンプレートにパラメーターを渡すことだけです。次にcompositionフラグの中で、下記のような単純なテキスト（タイトルなど）を渡します。

```
<ui:define name="title">CD form</ui:define>
```

あるいは、下記のようなコンポーネント・ツリー全体を渡します。

```
<ui:define name="content">
<!-- use the form tag to setup this form -->
<h:form id="cdForm">
...
...
...
</h:form>
</ui:define>
```

定義したり渡したりできる論理領域は数多くありますが、cdForm.xhtmlは、内容とタイトルという2つの論理領域しか渡していないことに注意してください。

タイトルとFacelets

この記事には3つのサンプルが付属しています。最初のサンプルはタイルを使用し、2番目のサンプルはFaceletsを使用し、3番目は構成コンポーネントを使用しています。タイルの例を含めてある理由は、2つのフレームワークでのテンプレートに関する手法を比較対照できるようにするためです。皆さん自身で試してみて、違いを実感してください。

構成コンポーネント

テンプレートを定義し、使用するためにだけFaceletsを使うのであれば、皆さんは少しがっかりするかも知れません。確かにFaceletsのテンプレートは完全な機能を備えており、また表現力も豊かですが、フレームワークとしてはそれほど多くの機能を持っていません。（タイルのようなフレームワークは、デフォルトや関連テンプレートの階層構造などを定義するために適しています。）

ただしFaceletsの本来の実力は、テンプレートで発揮されるのではなく、構成コンポーネントでこそ発揮されるのです。（面白いことに、構成コンポーネントはFaceletsのテンプレート動作にとっても有利なのです。例えば、Faceletsの中ではすべてがコンポーネント・ツリーの中のコンポーネントとして扱われるため、f:verbatimタグや様々なh:outputTextタグを省略できます。これについては後ほど詳しく説明します。）

この記事のこれから先では、構成コンポーネントの作成や使い方に関わる手順に注目して行きます。その前に、なぜこれがそれほど素晴らしいのかを確実に理解することにしましょう。

DRYの原則を破る

皆さんは、リスト3に示すようなコード断片を書いたことがあるでしょうか。

リスト3. 構成コンポーネントを使う前の姿

```
<h:dataTable id="items" value="#{CDManagerBean.cds}" var="cd"
```

```
 rowClasses="oddRow, evenRow" headerClass="tableHeader">

<!--  Title -->
<h:column>
 <f:facet name="header">
  <h:panelGroup>
   <h:outputText value="Title" />

                <f:verbatim>[</f:verbatim>

   <h:commandLink styleClass="smallLink"
                              action="#{CDManagerBean.sort}">
    <h:outputText id="ascTitle" value="asc" />
    <f:param name="by" value="title"/>
    <f:param name="order" value="asc"/>
   </h:commandLink>

   <h:outputText value="," />
   <!-- Sort descending -->
   <h:commandLink styleClass="smallLink"
                         action="#{CDManagerBean.sort}">
    <h:outputText id="decTitle" value="dec" />
    <f:param name="by" value="title"/>
    <f:param name="order" value="dec"/>
   </h:commandLink>
        <f:verbatim>]</f:verbatim>
  </h:panelGroup>
 </f:facet>

 <h:outputText value="#{cd.title}" />
</h:column>

<!--  Artist -->
<h:column>
 <f:facet name="header">
  <h:panelGroup>
   <h:outputText value="Artist" />
      <f:verbatim>[</f:verbatim>

        <h:commandLink styleClass="smallLink"
                           action="#{CDManagerBean.sort}">
        <h:outputText id="ascArtist" value="asc" />
    <f:param name="by" value="artist"/>
    <f:param name="order" value="asc"/>
      </h:commandLink>

   <h:outputText value="," />
     <!-- Sort descending -->
   <h:commandLink styleClass="smallLink"
                action="#{CDManagerBean.sort}">
    <h:outputText id="decArtist" value="dec" />
    <f:param name="by" value="artist"/>
    <f:param name="order" value="dec"/>
   </h:commandLink>
        <f:verbatim>]</f:verbatim>
  </h:panelGroup>
 </f:facet>
 <h:outputText value="#{cd.artist}" />
</h:column>
```

この、listing.xhtmlからのコードは、カラム・ヘッダーを生成し、またサンプル・アプリケーションのListingページに対して、昇順と降順でソートしたリンクを生成します。複数のカラムを出力するために、幾つもの場所でコードを重複せざるを得なかったことに注意してください。（ また

皆さんは上の例で、私が${..} と#{..} とを切り換えていることにも気が付くかも知れません。紛らわしいですが、これらは同じことをしているのです。 ）

TitleカラムとArtistカラムを描画するために繰り返されているコードは、DRY、つまり『don't repeat yourself (繰り返すな) 』の原則を破っています。ところで、こうすることの不都合は一体何なのでしょう。例えば、リスティングの中に平均5つのカラムがあり、アプリケーションの中には別々な20のリスティングがあるとしましょう。リスト3の手法を使うと、同じ35ラインのコードを100回繰り返さなければならず、合計すると3,500ラインのコードになってしまうのです。これだけのコードを維持管理することは大変なことです。では、リスティングのプレゼンテーションを変更することに決めとしたら、あるいは、 (ここで息継ぎをしましょう！) リストをフィルターにかけるための汎用の方法を追加するとしたらどうでしょう。桁外れの大仕事になってしまいます。

では、リスト3を下記と比べてみてください。

リスト4. フィールドを作成するための新しい方法

```
<h:dataTable id="items" value="#{CDManagerBean.cds}" var="cd"
  rowClasses="oddRow, evenRow" headerClass="tableHeader">

 <a:column entity="${cd}" fieldName="title" backingBean="${CDManagerBean}"/>
 <a:column entity="${cd}" fieldName="artist" backingBean="${CDManagerBean}"/>
```

どうやら、70ライン以上ものコードを、たった4ラインで置き換えられたようです。ご想像の通り、a:columnが構成コンポーネントなのです。Faceletsでは、このようなコンポーネントの定義が簡単にできるのです (リスト5) 。

リスト5. ソート・リンクを持つカラムをcolumn.xhtmlで作成する

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:z="http://www.qualcomm.com/jsf/core"
      xmlns:c="http://java.sun.com/jstl/core"
      xmlns:fn="http://java.sun.com/jsp/jstl/functions">

THIS TEXT WILL BE REMOVED
<ui:composition>
 <!--  The label attribute is optional. Generate it if it is missing. -->
 <c:if test="${empty label}">
     <c:set var="label" value="${fieldName}" />
 </c:if>

 <!--  The sort attribute is optional. Set it to true if it is missing. -->
 <c:if test="${empty sort}">
     <c:set var="sort" value="${true}" />
 </c:if>

 <h:column>
   <f:facet name="header">
```

```
     <h:panelGroup>
       ${label}
       <c:if test="${sort}">
           [
         <h:commandLink styleClass="smallLink"
       action="#{backingBean.sort}">
           <h:outputText value="asc" />
           <f:param name="by"
              value="${fieldName}"/>
           <f:param name="order" value="asc"/>
         </h:commandLink>
     ,
         <!-- Sort descending -->
         <h:commandLink styleClass="smallLink"
              action="#{backingBean.sort}">
           <h:outputText value="asc" />
           <f:param name="by"
              value="${fieldName}"/>
           <f:param name="order" value="dec"/>
         </h:commandLink>
     ]
       </c:if>
     </h:panelGroup>
   </f:facet>
   <!--  Display the field name -->
   <h:outputText value="${entity[fieldName]}"/>
        </h:column>
</ui:composition>
THIS TEXT WILL BE REMOVED AS WELL
</html>
```

注意すべき点

より高度な例に進む前に、幾つかのことに注目して欲しいと思います。まず、リスト5では、値バインディング（value binding）を下記のような汎用的な方法で参照していることに注意してください。

```
<h:outputText value="${entity[fieldName]}"/>
```

第2に、この構成コンポーネントを呼び出す際には、下記のようにentityとfieldNameを属性として渡しています。

```
<a:column entity="${cd}" fieldName="title" backingBean="${CDManagerBean}"/>
```

Faceletsが使用しているEL仕様では、ドット（．）表記、あるいは、もっと使用されることが少ないMap表現を使ってフィールドを参照することができます。例えば ${entity[fieldName]} は、上記のように呼び出された場合には、CDManager.titleと等価です。また、f:verbatimタグや、補助的なh:outputTextが必要ないことにも注意してください。どんなFaceletsページを書いても、これが言えるのです。FaceletsはJSFコンポーネント・ツリーをよく理解しており、そのコンポーネント・ツリーを構築することこそ、Faceletsの全目標なのです。これは、JSPやタイルを使わずFaceletsを使うことによる、もう一つの利点です。

『column.xhtml』という構成コンポーネントを一度書いてしまえば、それを他の多くの場所で使うことができます。一般的なルールとしては、DRYの原則を破りそうな時には構成コンポーネントを使うことを考える、と言えるのです。

コンポーネントを作成する

構成コンポーネントを、column.xhtmlの例を使って簡単に見てきました。今度はその作り方の過程を、順を追って見て行きましょう。そのステップは次の通りです。

1. Faceletsタグ・ライブラリーを作成します。
2. web.xmlの中でタグ・ライブラリーを宣言します。
3. 名前空間を使ってtagfileをインポートします。

ステップ1. Facelets tagfileを作成する

『tagfile』は、facelet_taglib_1_0.dtdに従うファイルです。tagfileは、概念的にはJSPでのTLDファイルと似ています。リスト6は、タグ・ライブラリー・ファイルの一例です。

リスト6. タグ・ライブラリー・ファイル・・・arcmind.taglib.xml

```
<?xml version="1.0"?>
<!DOCTYPE facelet-taglib PUBLIC
  "-//Sun Microsystems, Inc.//DTD Facelet Taglib 1.0//EN"
  "facelet-taglib_1_0.dtd">
<facelet-taglib>
      <namespace>http://www.arc-mind.com/jsf</namespace>
 <tag>
  <tag-name>field</tag-name>
  <source>field.xhtml</source>
 </tag>
 <tag>
  <tag-name>column</tag-name>
  <source>column.xhtml</source>
 </tag>
 <tag>
  <tag-name>columnCommand</tag-name>
  <source>columnCommand.xhtml</source>
 </tag>
</facelet-taglib>
```

arcmind.taglib.xmlファイルは3つのタグを宣言しています。つまりfieldタグと、columnタグ（これは既に見ました）、そしてcolumnCommandタグです。すべきことは、tag-nameを使ってタグ名を規定することと、実装ファイルの位置を規定することだけです。実装ファイルの名前は相対的です。これらのコードは（DTDを含めて）すべて、サンプルWebアプリケーションのWEB-INF\facelets\tagsファイルの中にあります。

訳註：円マーク（￥）は原文ではバックスラッシュです。

上記のタグ要素の前に宣言されているnamespace要素に注意してください。後で、このタグ・ライブラリーを別のFaceletsページから使うために、この要素が必要になります。

ステップ2. web.xmlの中でタグ・ライブラリーを宣言する

タグ・ライブラリーがあるのは素晴らしいことですが、それを有効にするためには、それが存在していることをFaceletsに伝える必要があります。それには、web.xmlファイルの中のfacelets.LIBRARIES initパラメーターを使います。これを下記に示します。

```
<context-param>
 <param-name>facelets.LIBRARIES</param-name>
 <param-value>
  /WEB-INF/facelets/tags/arcmind.taglib.xml
 </param-value>
</context-param>
```

セミコロン区切りのリストとしてfacelets.LIBRARIESを渡すことによって、幾つでもtagfileを定義することができます。

ステップ3. 名前空間を使ってtagfileをインポートする

tagfileを作成し、Faceletsのタグ・ライブラリーの中で定義してしまえば、使用準備完了です。tagfileを使うには、XML名前空間として宣言します。これを下記に示します。

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:a="http://www.arc-mind.com/jsf">

...
...

<a:column entity="${cd}" fieldName="title"
  backingBean="${CDManagerBean}"/>
<a:column entity="${cd}" fieldName="artist"
  backingBean="${CDManagerBean}"/>
<a:column entity="${cd}" fieldName="price"
  backingBean="${CDManagerBean}" sort="${false}"/>
<a:columnCommand label="Edit" action="editCD"

              backingBean="${CDManagerBean}"/>
```

下記のように定義された名前空間に注意してください。

```
xmlns:a="http://www.arc-mind.com/jsf"
```

名前空間の値は、ステップ1のタグ・ライブラリーの中で宣言した名前空間要素の値と同じです。

高度なトリックとヒント

これで、構成コンポーネントの基本は説明しました。ここまでで得られた知識を活用すれば、再利用可能なコンポーネントを作成することができます。追加として、私がFaceletsを使う中で見つけた、ちょっとしたトリックを紹介しましょう。こうしたトリックを使うと、構成コンポーネントが一層便利になり、また場合によっては小さな問題に対処できるのです。例えば、下記に示すような、cdForm.xhtmlテンプレートからのコード断片を考えてみてください。

リスト7. cdForm.xhtmlからのコード断片

```
<h:form id="cdForm">

  <h:inputHidden id="cdid" value="#{CDManagerBean.cd.id}" />
```

```
 <h:panelGrid id="formGrid" columns="3" rowClasses="row1, row2">

<!-- Title                                    -->
<h:outputLabel id="titleLabel" for="title" styleClass="label"
                    value="Title" />
<h:inputText id="title" value="#{CDManagerBean.cd.title}"
        required="true" />
<h:message id="titleMessage" for="title" styleClass="errorText"/>

<!-- Artist                                    -->
<h:outputLabel  id="artistLabel" for="artist" styleClass="label"
                    value="Artist" />
<h:inputText id="artist" value="#{CDManagerBean.cd.artist}"
        required="true" />
<h:message id="titleMessage" for="artist"
                    styleClass="errorText"/>

<!-- Price                                    -->
<h:outputLabel  id="priceLabel" for="price" styleClass="label" value="Price" />
<h:inputText id="price" value="#{CDManagerBean.cd.price}"
        required="true">
 <f:validateDoubleRange minimum="15.0" maximum="100.0" />
</h:inputText>
<h:message id="priceMessage" for="price" styleClass="errorText"/>
```

上記のページは、概念的にはリスト3で示したページと似ています。つまり、少しばかりFaceletsに愛される余地があり、またフィールド構成コンポーネントで重複コードを排除できそうです。このコードを元に、フィールドを表示するための構成コンポーネントを作成するのは容易なはずですが、1つだけ小さな問題があります。皆さんには、その問題が分かるでしょうか。フォームのpriceフィールドを見ると、バリデーターが含まれているのです。

では、どうやってバリデーターを構成コンポーネントに渡すのでしょう。

サブ要素を渡す

ここに、Faceletsの汚れた秘密があるのです。つまり構成コンポーネントは基本的に、一種のテンプレートです。従って、ある特定なui:insertに付随するui:defineタグを使ってテンプレート引数を渡すことができるのです。あるいはボディーを、デフォルトのui:insertとして渡すことができます。

リスト8は、正にそのようにフィールド・コンポーネント（field. xhtml）を実装した例です。

リスト8. field.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:z="http://www.qualcomm.com/jsf/core"
      xmlns:c="http://java.sun.com/jstl/core"
      xmlns:fn="http://java.sun.com/jsp/jstl/functions"
      xmlns:t="http://myfaces.apache.org/tomahawk">
```

```
THIS TEXT WILL BE REMOVED
<ui:composition>

      <!--  The label is optional.
               Generate it if it is missing. -->
  <c:if test="${empty label}">
   <c:set var="label" value="${fieldName}" />
  </c:if>

   <!-- The required attribute is optional,
                initialize it to true if not found. -->
  <c:if test="${empty required}">
   <c:set var="required" value="true" />
  </c:if>


  <h:outputLabel id="${fieldName}Label"
                      value="${label}" for="#{fieldName}" />

  <h:inputText id="#{fieldName}" value="#{entity[fieldName]}"
               required="${required}">
     <ui:insert />
  </h:inputText>

  <!--  Display any error message that are found -->
  <h:message id="${fieldName}Message"
   style="color: red; text-decoration: overline"
   for="#{fieldName}" />

</ui:composition>
THIS TEXT WILL BE REMOVED AS WELL

</html>
```

ここまで来れば、リスト8での回避策は、ほとんど皆さんが想像した通りのものになるはずです。h:inputTextの内側で、名前のないui:insertタグが使われていることに注意してください。これが手に入りさえすれば、この構成コンポーネントを使うことができます（リスト9）。

リスト9. フィールド・タグ構成コンポーネント

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:a="http://www.arc-mind.com/jsf">

...
<h:form id="cdForm">

 <!-- Title, Artist, Price -->
 <a:field fieldName="title" entity="#{CDManagerBean.cd}" />
 <a:field fieldName="artist" entity="#{CDManagerBean.cd}" />
 <a:field fieldName="price" entity="#{CDManagerBean.cd}" >
  <f:validateDoubleRange minimum="15.0" maximum="100.0" />
 </a:field>
```

price用のフィールド・タグには、匿名インサート(anonymous insert)としてバリデーターが渡されます。他のフィールドはボディーを定義していないため、匿名インサートによってデフォルトで追加されるものは何もありません。

アクションを渡す

場合によると、ツールバーやナビゲーション・リストのような要素を作成するために、アクション・バインディングを渡したいことがあります。しかし標準的な表現言語では、そうしたことはできません。しかし、ここに別の手段があるのです。オブジェクトからフィールドを参照するのと同じ方法で、オブジェクトの中のメソッドを参照できるのです。ですから、アクション・バインディングを作成するコンポーネントを作成するためには、次のようにすればよいのです（ columnCommand.xhtmlより ）。

```
<h:commandLink id="#{action}" value="#{label}"
                              action="#{backingBean[action]}"/>
```

アクション属性のvalueをよく調べてみましょう。先ほど実体からフィールドを参照した方法と同じ方法を使って、メソッドにアクセスしていることに注意してください。このコンポーネントは、次の構文を使って呼び出すことができます。

```
<a:columnCommand label="Edit" action="editCD"
backingBean="${CDManagerBean}"/>
```

このコールは呼び出されると、CDManagerBeanからのeditCD() メソッドをリンクにバインドします。リスト10は、columnCommand.xhtmlの完全なリストを示しています。

リスト10. columnCommand.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:z="http://www.qualcomm.com/jsf/core"
      xmlns:c="http://java.sun.com/jstl/core">

THIS TEXT WILL BE REMOVED
<ui:composition>

 <!--  The label is optional. Generate it if it is missing. -->
 <c:if test="${empty label}">
   <c:set var="label" value="${action}" />
 </c:if>

 <h:column>
  <f:facet name="header">
   <h:panelGroup>
    <h:outputText value="#{label}" />
   </h:panelGroup>
  </f:facet>
  <h:commandLink id="#{action}" value="#{label}"
```

```
                                    action="#{backingBean[action]}"/>
 </h:column>

</ui:composition>
THIS TEXT WILL BE REMOVED AS WELL

</html>
```

Faceletsの欠点

ここまでで、Faceletsを使うことによる利点、つまり『共通語』が（サーブレット出力ではなく）コンポーネントであるコンポーネント構成とテンプレート・フレームワークについて、明確に示せたと思います。しかし、Faceletsを採用することには多少問題もあります。1つは、Faceletsに対するIDEのサポートがほとんどない、という点です。FaceletsをサポートするEclipse IDE実装は、1つしかありません（これは商用のものです。参考文献を見てください。）しかも、コード完了（code completion）はサポートされていないようです。

また、FaceletsをデバッグするためのIDEサポート（つまりブレークポイントの設定など）もありません。デバッグに関しては、Faceletsのマニュアルを読み、Faceletsの（JDK 1.4風の）ログをオンにし、その開発に合わせてinitパラメーターを適切に設定する他ありません。

良い点としては、Facelets APIは非常に自然で直感的なことです。デバッグは、最初は少しばかり秘密がかっていますが、何とかなるものです。Faceletsディストリビューションに同梱されているデモ・アプリケーションにはカスタム・タグやファンクションの例はありませんが、コア・プロジェクト・コードの例はあるので、これをガイドとして使うことができます。

新しいJSFコンポーネント・ライブラリーを使う場合には、そのライブラリーを公開するFaceletsタグ・ライブラリー・ファイルを持っている必要があります。OracleやTomahawkなどのようなメジャーなコンポーネント・ライブラリー用のタグ・ライブラリーは幾つかありますが、そうしたものでさえ、多少は手を加える必要があります。私は、自分のアプリケーションの中にTomahawkのカレンダー・コンポーネントを組み込むために、Tomahawkタグ・ライブラリーを小細工しなければなりませんでした。確かにコンポーネントをエクスポートするタグ・ライブラリー・ファイルを書くのは容易ですが、余計な手間であることは事実です。皆さんも新しいカスタム・コンポーネント・ライブラリーを使いたい場合には、タグ・ライブラリー・ファイルを書く必要があります。

またFaceletsは、他の実装での問題のため、MyFaces 1.1.1とSunの1.2 JSF基準実装（RI: reference implementation）でしか動作しないようです（SunのJSF RI 1.2はまだ正式には世に出ていません）。1.1 RIでFaceletsを使うことはできません。IBM WebSphereでMyFacesを使うことはできますが、FaceletsをIBMの実装で使うことはできません。（最新版のFaceletsを使う場合には、まだ世に出ていない闇版のMyFaces 1.1.2を使う必要があります。）

また、MyFaces 1.1の基礎となる機構とJSF RI 1.2の基礎となる機構は異なることにも注意してください。ただしFaceletsは、両者の現在の実装（つまり今は闇版であるMyFaces 1.1.2 とJSF RI 1.2）に対応しようとしており、Faceletsの開発作業の大部分はそのために費やされているようです。両者が、もう少し近づき、堅牢になれば素晴らしいことです。そうすれば、どちらの環境でもFacelets

が同じ動作をするように細工するための時間を減らすことができ、Facelets自体を改善するためにもっと時間をかけられるはずなのです。

最後に

Faceletsには少しばかり問題はありますが、私としては皆さんがすぐにFaceletsをダウンロードし、使い始めるように強く勧めたいと思います。FaceletsはJSFの将来であり、そうあるべきです。Faceletsを使うことによって、どんなJSFの嵐の中でもDRYを維持することができます。もし現在のプロジェクトではFaceletsを使えないのであれば、次のプロジェクトのために頭の中に置いておいてください。

私はこれまでFaceletsを使って、構成コンポーネントやカスタムFaceletsタグ、内部CRUDフレームワーク用のファンクションなど様々なライブラリーを作ってきました。今回のような入門的な記事では紹介できませんでしたが、私は構成コンポーネントを構築するための様々なトリックや手法も見つけています (例えば、チェックボックスを自動生成するフィールド・タグや、カレンダー・コンポーネント、コンポーネントにバインドされたタイプの値バインディングに基づくテキスト・フィールドなど。) しかしこの記事では、そうしたものを紹介するよりも、構成コンポーネントを作成し、動作させるための基本に焦点を絞りました。ここで学んだことを利用すれば、ほんの少しのカスタム・ファンクションとカスタムFaceletsタグのみを使って、驚くようなコンポーネントを作ることができるでしょう。

謝辞

この記事を見直してくださり、また助言を頂いた、Faceletsの作者であるJacob Hookomに深く感謝致します。また、注意深く洞察に満ちた編集を行ってくれたAthenzに感謝します。

ダウンロード

内容	ファイル名	サイズ
Facelets source code	j-facelets_code.zip	267KB
Facelets source code with jars and wars	j-faceletsjarsandwars.zip	47MB

著者について

Richard Hightower

Rick Hightowerは、JSFやSpring、Hibernateなどに関するトレーニングの専門会社であるArcMind Incの最高技術責任者です。J2EE開発への極限プログラミングの応用を解説した人気の著書『Java Tools for Extreme Programming』の共著者であり、また『Professional Struts』の共著者でもあります。