

# JSF 2 の魅力: 第 2 回 テンプレート機能と複合コンポーネント

## JavaServer Faces 2 を使って拡張可能な UI を実装する

David Geary

President

Clarity Training, Inc.

2009年 6月 02日

JSF (Java™Server Faces) 2 では、2 つの強力な機能によって簡単に変更および拡張ができるユーザー・インターフェースを実装することができます。その 2 つの機能とは、テンプレート機能と複合コンポーネントです。JSF 2 の新機能を紹介する 3 回連載の第 2 回目となる今回の記事では、JSF 2.0 Expert Group のメンバー、David Geary が Web アプリケーションでテンプレート機能と複合コンポーネントを最大限に活用する方法を紹介します。

[このシリーズの他の記事を見る](#)

Struts という新しい Web フレームワークに取り組んでいる Craig McClanahan と知り合ったのは、私が JSP (JavaServer Pages) メーリング・リストで盛んに活動していた 2000年のことです。Swing からサーバー・サイドの Java プログラミングに移行していた当時、私は Swing のレイアウト・マネージャーと同じような考え方で、小さなフレームワークを実装して JSP ビューのレイアウトをコンテンツから切り離していました。この私のテンプレート・ライブラリーを Struts に組み込む気はあるかと Craig が聞いてきたので、私はその申し出を快諾しました。その結果 Struts 1.0 にバンドルされた Struts Template Library は Struts の著名な Tiles ライブラリーのベースとなり、Tiles ライブラリーは最終的にトップ・レベルの Apache フレームワークとなりました。

現在、JSF 2 のデフォルト表示技術となっている Facelets は、その大部分が Tiles に基づいたテンプレート・フレームワークです。さらに、JSF 2 には複合コンポーネントという強力なメカニズムも用意されています。複合コンポーネントは Facelets のテンプレート機能をベースにしているため、Java コードを作成したり XML による構成を行ったりしなくても、カスタム・コンポーネントを実装することができます。この記事ではテンプレート機能と複合コンポーネントについて、JSF 2 を最大限に利用するための以下の 3 つのヒントに沿って紹介します。

- ヒント 1: DRY を守る
- ヒント 2: 構成することを心掛ける
- ヒント 3: LEGO のように考える

## Facelets と JSF 2

JSF 2.0 Expert Group は、オープンソースの Facelets 実装を標準化するなかで、そのベースである Facelets API にいくつかの変更を加えました。しかし、タグ・ライブラリーとの後方互換性はそのまま維持しました。したがって、Facelets のオープンソース・バージョンで実装された既存のビューは JSF 2 でも有効に機能します。

Facelets に備わった多くの機能についての詳細は、Rick Hightower の 2 つの記事、「[Facelets はぴったりと JSF にフィットします](#)」と「[高度な Facelets プログラミング](#)」を参照してください。

## ヒント 1: DRY を守る

ソフトウェア開発者として私が初めて手掛けた仕事は、UNIX® ベースの CAD/CAM (Computer-Aided Design and Computer-Aided Manufacturing) システムに GUI を実装することでした。

初めはそこそこ順調に進んでいたのですが、次第に私の作成したコードが問題となってきました。リリースする頃までにはバグ修正の恐怖におびえるほどシステムが不安定になり、結局はリリースによってバグ・レポートが次々と寄せられることになりました。

もし、私があのプロジェクトで DRY (Don't Repeat Yourself) 原則に従っていたとしたら、あれほどの面倒にはなっていなかったはずです。DRY 原則とは Dave Thomas と Andy Hunt が作り出した造語で（「[参考文献](#)」を参照）、以下の原則を提唱しています。

「あらゆる情報は、システム内で他の情報と重複することなく、一義的で信頼すべき表現にすること。」

私の CAD/CAM アプリケーションは DRY 原則に従ってなく、コードの内容があまりにも相互に関連していたため、1 箇所を変更を加えると、どこか他の場所でも予期しない変更を引き起こす結果となったのです。

JSF 1 はいくつかの点で DRY 原則に反していました。例えば、管理対象 Bean の表現は 2 つ指定しなければなりません (1 つは XML、もう 1 つは Java コード)。複数の表現が必要になることから、管理対象 Bean は作成するにも、変更するにも困難でした。[第 1 回](#)で説明したように、JSF 2 では XML の代わりにアノテーションを使って管理対象 Bean を構成できるので、管理対象 Bean に唯一の信頼できる表現を指定することができます。

管理対象 Bean とは別に、すべてのビューに同じスタイルシートを含めるといった害のなさそうなプラクティスでさえも DRY 原則に違反し、驚くような問題を引き起こす可能性があります。例えばスタイルシートの名前を変更すると、複数のビューも変更せざるを得なくなります。それよりも、できることならスタイルシートのインクルードをカプセル化するに越したことはありません。

DRY 原則はコードの設計にも当てはまります。例えば、ツリーをたどるコードが複数のメソッドすべてに含まれている場合、ツリー・ウォークのアルゴリズムを (おそらくサブクラスに) カプセル化するのが得策なはずです。

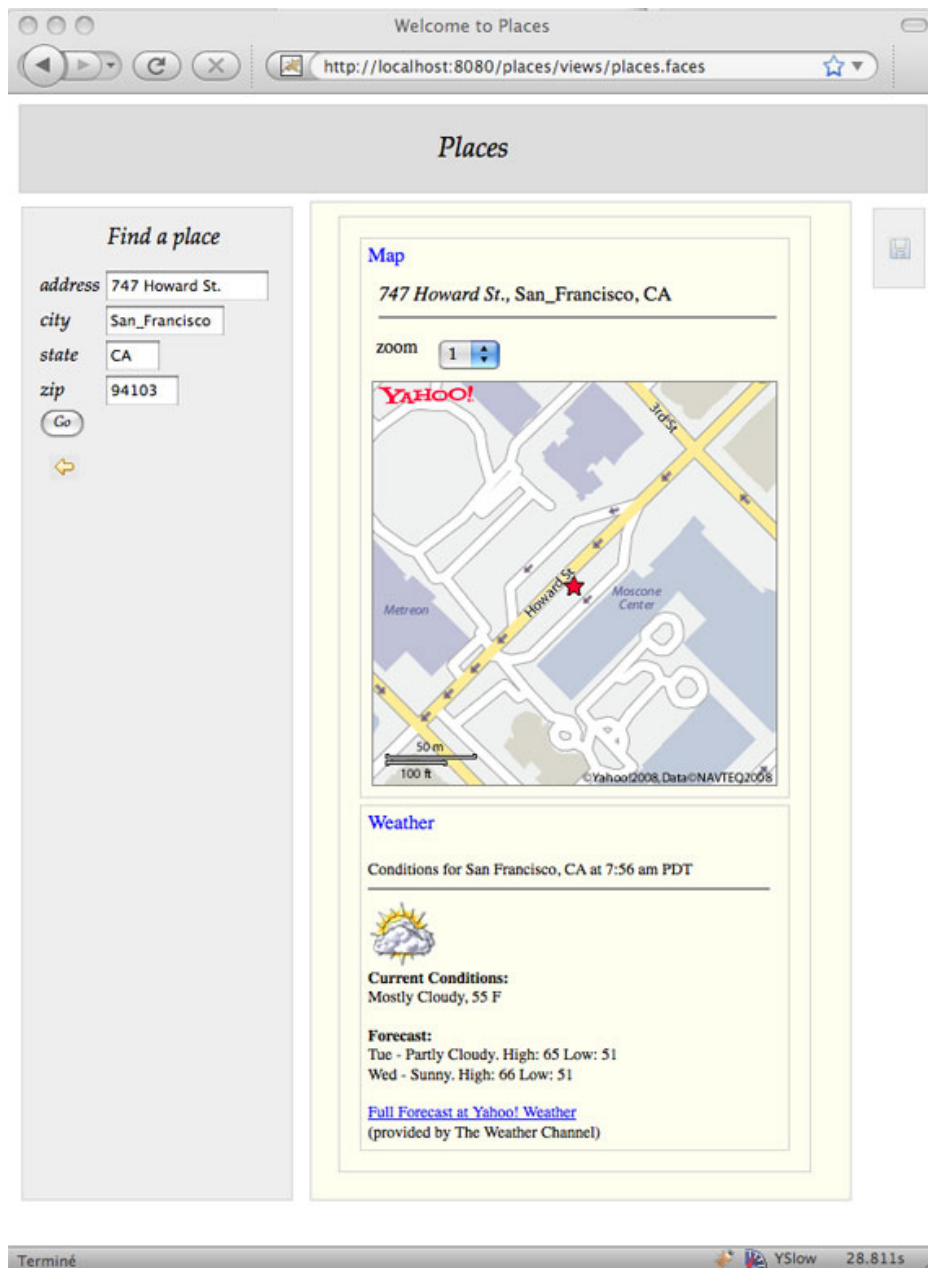
DRY 原則に従うことは、特に UI を実装するときに重要となります。開発中のほとんどの変更は、ほぼ間違いなく UI で発生するためです。

## JSF 2 のテンプレート機能

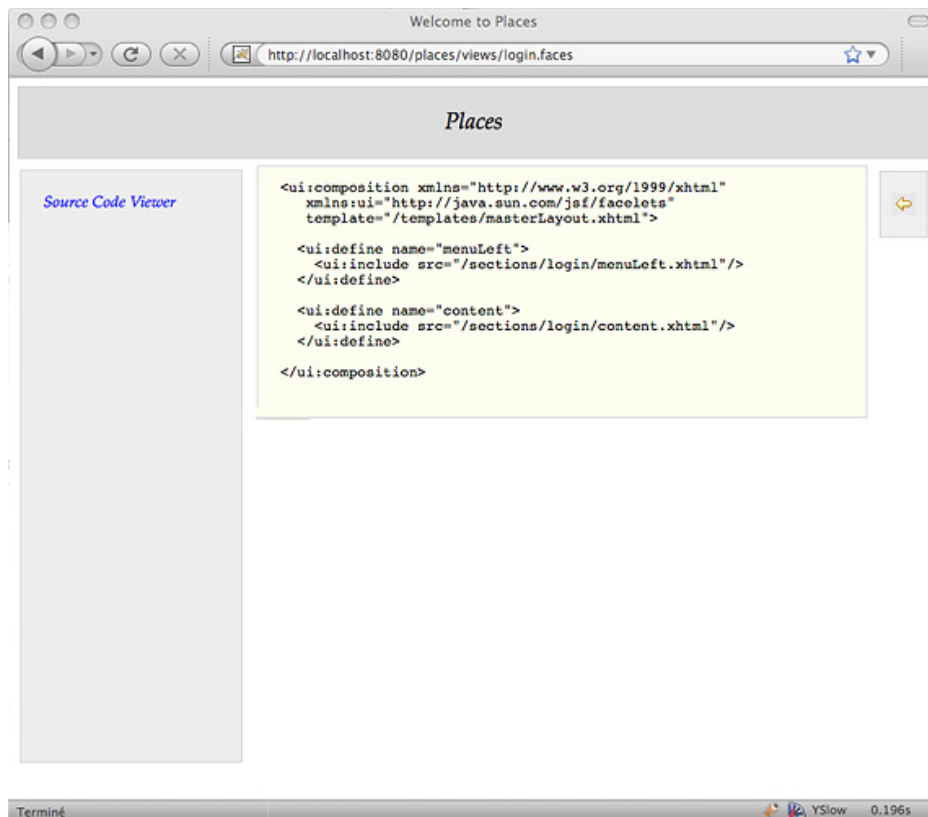
JSF 2 は DRY 原則をさまざまな方法でサポートします。そのうちの 1 つはテンプレート機能です。テンプレートはアプリケーションのビューに共通する機能をカプセル化します。したがって、その機能を指定するのは 1 度だけで済みます。JSF 2 アプリケーションではビューを作成するために、1 つのテンプレートが複数のコンポジション (composition) で使用されます。

第 1 回で導入した Places アプリケーションには、図 1 に示す 3 つのビューがあります。

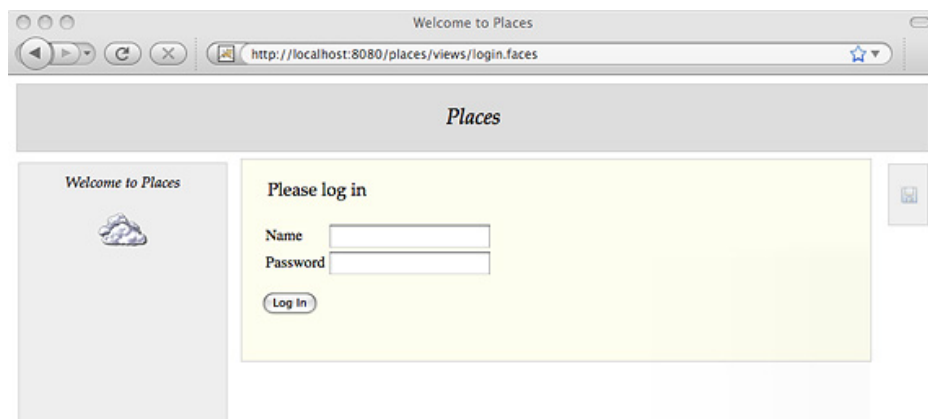
図 1a. Places アプリケーションのビュー: プレース、ソース・ビューアー、ログイン



## 図 1b. Places アプリケーションのビュー: プレース、ソース・ビューアー、ログイン



## 図 1c. Places アプリケーションのビュー: プレース、ソース・ビューアー、ログイン



多くの Web アプリケーションと同じく、Places アプリケーションに含まれる複数のビューは同じレイアウトを共有します。JSF のテンプレート機能を使えば、そのような共通のレイアウトを他の共有成果物 (JavaScript や CSS (Cascading Style Sheets) など) と併せてテンプレートにカプセル化することができます。リスト 1 は、[図 1](#) に示した 3 つのビューのテンプレートです。

### リスト 1. Places テンプレート: /templates/masterLayout.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:h="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/facelets">

<h:head>
  <title>
    <ui:insert name="windowTitle">
      #{msgs.placesWindowTitle}
    </ui:insert>
  </title>
</h:head>

<h:body>
  <h:outputScript library="javascript" name="util.js" target="head"/>
  <h:outputStylesheet library="css" name="styles.css" target="body"/>

  <div class="pageHeading">
    <ui:insert name="heading">
      #{msgs.placesHeading}
    </ui:insert>
  </div>

  <div class="menuAndContent">
    <div class="menuLeft">
      <ui:insert name="menuLeft"/>
    </div>

    <div class="content" style="display: #{places.showContent}">
      <ui:insert name="content"/>
    </div>

    <div class="menuRight">
      <ui:insert name="menuRight">
        <ui:include src="/sections/shared/sourceViewer.xhtml"/>
      </ui:insert>
    </div>
  </div>
</h:body>
</html>
```

**リスト 1** に記載したテンプレートは、このアプリケーションのすべてのビューのベースとなる部分 (以下に挙げます) を提供します。

- HTML の `<head>`、`<body>`、および `<title>`
- デフォルト・タイトル (このテンプレートを使用するコンポジションによって変更可能)
- CSS スタイルシート
- ユーティリティー JavaScript
- `<div>` とそれに対応する CSS クラスという形でのレイアウト
- 見出しのデフォルト・コンテンツ (変更可能)
- 右側メニューのデフォルト・コンテンツ (変更可能)

**リスト 1** に示されているとおり、テンプレートでは `<ui:insert>` タグを使ってレイアウトにコンテンツを挿入します。

**リスト 1** のウィンドウ・タイトル、見出し、右側メニューのところで行っているように、`<ui:insert>` タグの内容を指定すると、JSF はその指定された内容をデフォルト・コンテンツとして使用します。ログイン・ビューのマークアップを記載したリスト 2 を見るとわかるように、テンプレートを使用するコンポジションは、`<ui:define>` タグを使用してコンテンツを定義することも、デフォルト・コンテンツを変更することもできます。

## リスト 2. ログイン・ビュー

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  template="/templates/masterLayout.xhtml">

  <ui:define name="menuLeft">
    <ui:include src="/sections/login/menuLeft.xhtml"/>
  </ui:define>

  <ui:define name="content">
    <ui:include src="/sections/login/content.xhtml"/>
  </ui:define>

</ui:composition>
```

ログイン・ビューでは、ウィンドウのタイトル、見出し、右側メニューにテンプレートのデフォルト・コンテンツを使用します。ここで定義しているのはログイン・ビューに特有の機能、つまりコンテンツのセクションと左側メニューだけです。

ウィンドウ・タイトル、見出し、あるいは右側メニューに `<ui:define>` を指定すれば、テンプレートが定義するデフォルトのコンテンツを変更することもできます。その例として、リスト 3 のソース・ビューアー・ビュー (図 1 の真ん中のビュー) を見てください。

## リスト 3. ソース・ビューアー・ビュー

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  template="/templates/masterLayout.xhtml">

  <ui:define name="content">
    <ui:include src="/sections/showSource/content.xhtml"/>
  </ui:define>

  <ui:define name="menuLeft">
    <ui:include src="/sections/showSource/menuLeft.xhtml"/>
  </ui:define>

  <ui:define name="menuRight">
    <ui:include src="/sections/showSource/menuRight.xhtml"/>
  </ui:define>

</ui:composition>
```

ソース・ビューアー・ビューは、コンテンツ・セクションと右側メニューのコンテンツを定義しています。一方、左側メニューについては、リスト 1 のテンプレートが定義しているデフォルト・コンテンツを変更しています。

リスト 4 に、プレース・ビュー (図 1 の一番下のビュー) を記載します。



## リスト 4. プレース・ビュー

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  template="/templates/masterLayout.xhtml">

  <ui:define name="menuLeft">
    <ui:include src="/sections/places/menuLeft.xhtml"/>
  </ui:define>

  <ui:define name="content">
    <ui:include src="/sections/places/content.xhtml"/>
  </ui:define>

</ui:composition>
```

### JSF 2 のテンプレート機能

テンプレート機能の概念は単純なもので、複数のビューに共通の機能をカプセル化する単一のテンプレートを定義するというものです。それぞれのビューはコンポジションとテンプレートからなります。

JSF がビューを作成するときには、まずコンポジションのテンプレートをロードし、ロードしたテンプレートにそのコンポジションで定義するコンテンツを挿入します。

リスト 2、3、4 の類似性に注目してください。3 つのビューはいずれもテンプレートを指定した上で、コンテンツを定義しています。また、ビューのベースとなる部分の大半はテンプレートにカプセル化されたインクルード・ファイルとなっているため、新しいビューを容易に作成できることも注目の点です。

JSF のテンプレート機能を使用する上で、もう 1 つ興味深い点があります。それは、リスト 2、3、4 のようなビューは時間が経ってもそれほど変更されないため、ビュー・コードのかなりの部分は基本的に保守の必要がないことです。

テンプレートを使用するビューと同じく、テンプレートもほとんど変わることはありません。非常に多くの共通機能がほとんど保守の要らないコードにカプセル化されているため、開発者ビューの実際のコンテンツ (ログイン・ページの左側メニューの内容など) に専念することができます。このビューの実際のコンテンツに専念することこそが、次のヒントです。

## ヒント 2: 構成することを心掛ける

私の CAD/CAM GUI がリリースされてから間もなく、これとは別のプロジェクトに数ヶ月間、Bob という開発者と取り組みました。彼のコード・ベースで作業していたのですが、驚いたことに、コードを変更するのも、バグを修正するのも容易なことでした。

Bob のコードと私のコードとのたった 1 つの、そして最大の違いにはすぐに気付きました。それは、彼が作成したメソッドは概して 5 行から 15 行くらいしかない小さなメソッドであるということです。そしてこれらの小さなメソッドから、システム全体が組み立てられています。私は前のプロジェクトで、多くの内容が詰め込まれた長々としたメソッドを変更するのに四苦八苦していた一方、Bob はアトミックな機能の小さなメソッドを速やかに組み合わせていました。Bob のコードと私のコードを比較すると、保守のしやすさと拡張性は雲泥の差です。それに気付いてからというもの、私は小さなメソッドを信奉するようになりました。

その当時は Bob も私も気付いていませんでしたが、私たちが使用していたのは以下に示す Smalltalk の Composed Method というデザイン・パターンでした(「[参考文献](#)」を参照)。

「ソフトウェアを複数のメソッドに分割し、それぞれのメソッドが識別可能な 1 つのタスクを同じ抽象化レベルで実行するようにすること。」

Composed Method パターンの使用によってもたらされるメリットについては、十分に解説されているので (Neal Ford の記事「[進化するアーキテクチャーと新方式の設計: Composed Method と SLAP](#)」で詳しくてわかりやすい説明がされているので参照してください)、ここでは Composed Method パターンをどのように JSF ビューに適用できるかに焦点を絞ります。

JSF 2 では小さなビューのフラグメントを組み合わせてビューを構成するように奨励しています。テンプレート機能は共通の機能をカプセル化することによって、ビューを複数の部分に分割します。また、前のリストで説明した JSF 2 が提供する `<ui:include>` タグを使用することで、ビューをさらに小さな機能部分に分割することができます。一例として、図 2 に Places アプリケーションのログイン・ページに表示される左側メニューを示します。

## 図 2. ログイン・ページの左側メニュー



リスト 5 は、上記メニューのコンテンツを定義するファイルです。

## リスト 5. ログイン・ビューの左側メニューの実装

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

  <div class="menuLeftText">
    #{msgs.welcomeGreeting}

    <div class="welcomeImage">
      <h:graphicImage library="images" name="cloudy.gif"/>
    </div>
  </div>

</html>
```

リスト 5 のマークアップは単純なので、ファイルは読みやすく、理解しやすく、保守するにも拡張するにも難しいことはありません。これと同じコードが、ログイン・ビューの実装に必要なすべてのものを 1 つに詰め込んだ長々とした XHTML ページに埋もれているとしたら、変更するのはかなり厄介です。

図 3 に、プレース・ビューの左側メニューを示します。



## 図 3. プレース・ビューの左側メニュー

リスト 6 は、プレース・ビューの左側メニューの実装です。

## リスト 6. プレース・ビュー左側メニューの実装

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:util="http://java.sun.com/jsf/composite/components/util">

  <div class="placesSearchForm">
    <div class="placesSearchFormHeading">
      #{msgs.findAPlace}
    </div>

    <h:form prependId="false">
      <h:panelGrid columns="2">

        #{msgs.streetAddress}
        <h:inputText value="#{place.streetAddress}" size="15"/>

        #{msgs.city} <h:inputText value="#{place.city}" size="10"/>
        #{msgs.state} <h:inputText value="#{place.state}" size="3"/>
        #{msgs.zip} <h:inputText value="#{place.zip}" size="5"/>

        <h:commandButton value="#{msgs.goButtonText}"
          style="font-family:Palatino;font-style:italic"
          action="#{place.fetch}"/>

      </h:panelGrid>
    </h:form>
  </div>

  <util:icon image="#{resource['images:back.jpg']}"
    actionMethod="#{places.logout}"
    style="border: thin solid lightBlue"/>

</ui:composition>
```

フォームを実装するリスト 6 では、icon コンポーネントを使用しています (この icon コンポーネントについては、この後の「[icon コンポーネント](#)」で説明します。とりあえずは、ページの作成者が画像とメソッドをアイコンに関連付けられることを知っておけば十分です)。ログアウトのアイコンに使用する画像は、図 3 の下に示されています。このログアウト・アイコンのメソッド `places.logout()` はリスト 7 のとおりです。

## リスト 7. `Places.logout()` メソッド

```
package com.clarity;
```

```

...
@ManagedBean()
@SessionScoped

public class Places {
    private ArrayList<Place> places = null;
    ...
    private static SelectItem[] zoomLevelItems = {
        ...
    }
    public String logout() {
        FacesContext fc = FacesContext.getCurrentInstance();
        ELResolver elResolver = fc.getApplication().getELResolver();

        User user = (User)elResolver.getValue(
            fc.getELContext(), null, "user");

        user.setName("");
        user.setPassword("");

        setPlacesList(null);

        return "login";
    }
}

```

マークアップは 30 行を限度としている私としては、プレース・ビューの左側メニューを実装する [リスト 6](#) はコード過多のしきい値に近いと言えます。このリストは多少読みづらいので、コードの 2 つの内容、つまりフォームとアイコンをそれぞれ個別のファイルにリファクタリングするという手段を使えるはずです。リスト 8 に、[リスト 6](#) をリファクタリングしたバージョンを記載します。ここでは、フォームとアイコンが個別の XHTML ファイルにカプセル化されています。

## リスト 8. リファクタリングしたプレース・ビューの左側メニュー

```

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

    <div class="placesSearchForm">
        <div class="placesSearchFormHeading">
            #{msgs.findAPlace}
        </div>

        <ui:include src="addressForm.xhtml">
        <ui:include src="logoutIcon.xhtml">
        </div>

    </ui:composition>

```

リスト 9 に addressForm.xhtml を記載します。

## リスト 9. addressForm.xhtml

```

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

    <h:form prependId="false">
        <h:panelGrid columns="2">

            #{msgs.streetAddress}
            <h:inputText value="#{place.streetAddress}" size="15"/>

            #{msgs.city} <h:inputText value="#{place.city}" size="10"/>

        </h:panelGrid>
    </h:form>

```

```

    #{msgs.state} <h:inputText value="#{place.state}" size="3"/>
    #{msgs.zip} <h:inputText value="#{place.zip}" size="5"/>

    <h:commandButton value="#{msgs.goBackText}"
        style="font-family:Palatino;font-style:italic"
        action="#{place.fetch}"/>

</h:panelGrid>
</h:form>

</ui:composition>

```

リスト 10 に logoutIcon.xhtml を記載します。

## リスト 10. logoutIcon.xhtml

```

<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:util="http://java.sun.com/jsf/composite/components/util">

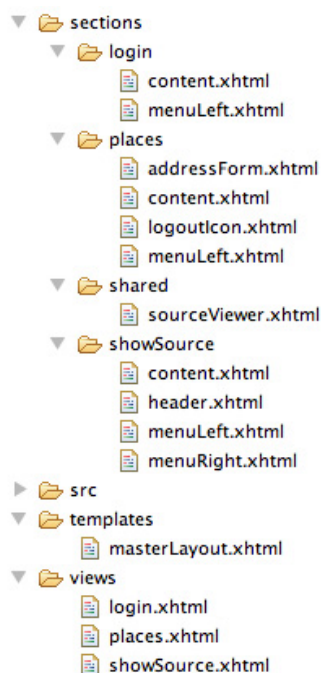
    <util:icon image="#{resource['images:back.jpg']}"
        actionMethod="#{places.logout}"
        style="border: thin solid lightBlue"/>

</ui:composition>

```

複数の小さなファイルからビューを構成するときに、Smalltalk の Composed Method パターンの成果が表れてきます。また、さらに変更に対応しやすいようなファイル構成にすることも可能です。その一例として、図 4 に Places アプリケーションの 3 つのビューを構成するファイルを示します。

## 図 4. Places アプリケーションのビュー



私が作成した 3 つのディレクトリー、sections、templates、views には、Places アプリケーションのビューを実装するために使用した XHTML ファイルのほとんどが格納されています。views ディ

レクトリーと templates ディレクトリーに置かれたファイルが変更されることはめったにないため、sections ディレクトリーに専念すればよいことになります。例えばログイン・ページの左側メニューに表示するアイコンを変更したいと思った場合には、sections/login/menuLeft.xhtml が変更の対象であるとすぐにわかります。

XHTML ファイルはどのようなディレクトリー構造で構成しても構いませんが、論理的な構造にすることによって、変更する必要のあるコードを見つけやすくなります。

DRY 原則に従うこと、そして Composed Method パターンを適用することに加え、カスタム・コンポーネントに機能をカプセル化するというのも賢い案です。コンポーネントは強力な再利用メカニズムなので、その威力を利用しない手はありません。JSF 1 とは異なり、JSF 2 ではカスタム・コンポーネントを簡単に実装できるようになっています。

## ヒント 3: LEGO のように考える

子供時代に私が好きだったおもちゃは2つあります。1つは化学実験用セット、そしてもう1つは LEGO です。どちらのおもちゃでもそうですが、基本的な構成要素を組み合わせることで何かを作り出すということが、生涯にわたって興味をそそるものになりました。それは、今ではソフトウェア開発という形になっています。

JSF の長所はこれまで常にそのコンポーネント・モデルにありましたが、この長所は今まで存分に実現されてはいませんでした。JSF 1 ではカスタム・コンポーネントの実装が困難だったからです。JSF 1 では Java コードを作成し、XML 構成を指定し、そして JSF のライフサイクルを十分に把握している必要がありました。JSF 2 では以下のようなカスタム・コンポーネントを実装することができます。

- 構成や XML などを必要としません。
- Java コードを必要としません。
- 開発者が機能を追加できます。
- 変更時のホット・デプロイメントをサポートします。

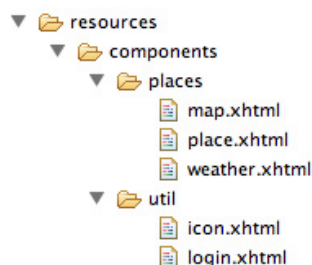
この記事の残りでは、Places アプリケーションを対象としたカスタム・コンポーネントとして、アイコン、ログイン・パネル、そしてある住所の地図とその地域の天気の情報を表示するパネル、の3つを実装する手順を紹介します。でもその前に、JSF 2 の複合コンポーネントの概要を説明しておきます。

## カスタム・コンポーネントの実装

JSF 2 は [Facelets のテンプレート機能](#)、リソース処理 ([第 1 回](#)で説明)、そして単純な命名規則を組み合わせて複合コンポーネントを実装します。複合コンポーネントとはその名前が示すとおり、既存の複数のコンポーネントを複合させて1つにしたコンポーネントのことです。

複合コンポーネントは、resources ディレクトリー配下の XHTML ファイルに実装し、単に慣例により名前空間とタグに関連付けます。Places アプリケーションの場合、私は複合コンポーネントを図 5 のように構成しています。

## 図 5. Places アプリケーションのコンポーネント



複合コンポーネントを使用するには、名前空間を宣言し、タグを使用します。名前空間は常に、`http://java.sun.com/jsf/composite` の後に、そのコンポーネントが常駐する (resources ディレクトリー配下の) ディレクトリー名が続いた形になります。複合コンポーネント自体の名前は、そのコンポーネントの XHTML ファイル名から `.xhtml` 拡張子を除いた名前となります。この規約のおかげで、構成は少しも必要ありません。例えば、Places アプリケーションで `login` コンポーネントを使うには以下のようにします。

```
<html xmlns="http://www.w3.org/1999/xhtml">
  ...
  xmlns:util="http://java.sun.com/jsf/composite/component/util">
  ...
  <util:login.../>
  ...
</html>
```

`icon` コンポーネントを使う場合は以下のようにします。

```
<html xmlns="http://www.w3.org/1999/xhtml">
  ...
  xmlns:util="http://java.sun.com/jsf/composite/components/util">
  ...
  <util:icon.../>
  ...
</html>
```

`place` コンポーネントは以下のようにして使用します。

```
<html xmlns="http://www.w3.org/1999/xhtml">
  ...
  xmlns:util="http://java.sun.com/jsf/composite/components/places">
  ...
  <places:place.../>
  ...
</html>
```

### **icon** コンポーネント: 単純な複合コンポーネント

Places アプリケーションでは図 6 に示す 2 つのアイコンを使用します。

#### 図 6a. Places アプリケーションのアイコン



## 図 6b. Places アプリケーションのアイコン



それぞれのアイコンはリンクです。ユーザーが図 6 の左側のアイコンをクリックすると、JSF は現在のビューのマークアップを表示します。右側のアイコンを起動するとユーザーをログアウトします。

リンクには CSS クラス名と画像を指定できるだけでなく、メソッドを接続することもできます。メソッドが関連付けられたリンクをユーザーがクリックすると、JSF はそのメソッドを呼び出します。

リスト 11 に、Places アプリケーションでマークアップを表示するために `icon` コンポーネントをどのように使用しているかを示します。

### リスト 11. `icon` コンポーネントによるマークアップの表示

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:util="http://java.sun.com/jsf/composite/components/util">

  <util:icon actionMethod="#{sourceViewer.showSource}"
            image="#{resource['images:disk-icon.jpg']}" />

  ...
</html>
```

リスト 12 は、`icon` コンポーネントを使用してログアウトする方法です。

### リスト 12. `icon` コンポーネントによるログアウト

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:util="http://java.sun.com/jsf/composite/components/util">

  <util:icon actionMethod="#{places.logout}"
            image="#{resource['images:back-arrow.jpg']}" />

  ...
</html>
```

リスト 13 に、`icon` コンポーネントのコードを記載します。

### リスト 13. `icon` コンポーネント

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:composite="http://java.sun.com/jsf/composite">

  <!-- INTERFACE -->
  <composite:interface>
    <composite:attribute name="image"/>
    <composite:attribute name="actionMethod"
      method-signature="java.lang.String action()"/>
  </composite:interface>

  <!-- IMPLEMENTATION -->
  <composite:implementation>
    <h:form>
```



```

<h:commandLink action="#{cc.attrs.actionMethod}" immediate="true">

<h:graphicImage value="#{cc.attrs.image}"
    styleClass="icon"/>

</h:commandLink>
</h:form>
</composite:implementation>
</html>

```

すべての複合コンポーネントと同じく、[リスト 13](#) の `icon` コンポーネントには 2 つのセクション、`<composite:interface>` と `<composite:implementation>` があります。`<composite:interface>` は、コンポーネントを構成するために使用できるインターフェースを定義するセクションです。`icon` コンポーネントには 2 つの属性があり、一方の `image` はコンポーネントの外観を定義し、もう一方の `actionMethod` はコンポーネントの振る舞いを定義します。

`<composite:implementation>` セクションにはコンポーネントの実装が含まれます。`<composite:implementation>` セクションでは `#{cc.attrs.ATTRIBUTE_NAME}` 式を使用して、コンポーネントのインターフェースに定義された属性にアクセスします (`cc` は JSF 2 式言語で予約されているキーワードで、複合コンポーネントを表します)。

[リスト 13](#) の `icon` コンポーネントは、このコンポーネントの画像に対する CSS クラスを `<h:graphicImage>` の `styleClass` 属性で指定していることに注目してください。この CSS クラスの名前は `icon` としてハードコーディングされているため、`icon` という名前の CSS クラスを定義すれば、JSF はアプリケーションのすべてのアイコンにそのクラスを使用することになります。しかし、別の CSS クラスを使用したい場合はどうすればよいのでしょうか。その場合は、インターフェースに CSS クラス用の属性を追加し、そこでデフォルト CSS クラスを設定することができます。このデフォルト CSS クラスは、実装で CSS クラスの属性が指定されないときに JSF が使用することになります。この属性の設定は、[リスト 14](#) のようになります。

## リスト 14. `icon` コンポーネント (リファクタリング後)

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml"
...
    xmlns:composite="http://java.sun.com/jsf/composite">

<composite:interface>
...
    <composite:attribute name="styleClass" default="icon" required="false"/>
...
</composite:interface>

<composite:implementation>
...
    <h:graphicImage value="#{cc.attrs.image}"
        styleClass="#{cc.attrs.styleClass}"/>
...
</composite:implementation>
</html>

```

[リスト 14](#) では、`icon` コンポーネントのインターフェースに `styleClass` という名前の属性を追加し、その属性をコンポーネントの実装で参照しています。このように変更したことで、アイコンの画像には以下のようにオプションの CSS クラスを指定できるようになりました。

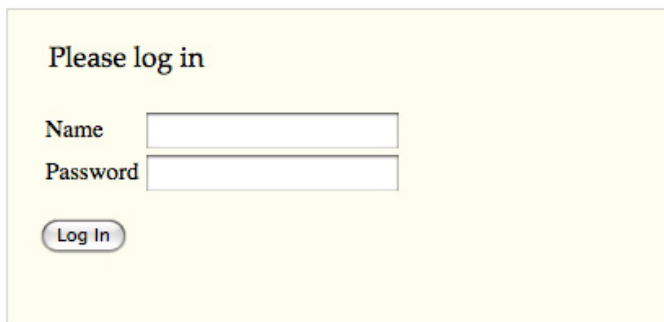
```
<util:icon actionMethod="#{places.logout}"
           image="#{resource['images:back-arrow.jpg']}"
           styleClass="customIconClass"/>
```

`styleClass` 属性を指定しなければ、JSF はデフォルト値である `icon` を使用します。

## login コンポーネント: 完全に構成可能なコンポーネント

JSF 2では、完全に構成可能な複合コンポーネントを実装することができます。例えば、Places アプリケーションには図 7 に示す `login` コンポーネントがあります。

### 図 7. Places アプリケーションの `login` コンポーネント



リスト 15 に、Places アプリケーションがこの `login` コンポーネントをどのように使用するかを示します。

### リスト 15. `login` コンポーネントの使用

```
<!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:comp="http://java.sun.com/jsf/composite/component/util">
  ...
  <util:login loginPrompt="#{msgs.loginPrompt}"
             namePrompt="#{msgs.namePrompt}"
             passwordPrompt="#{msgs.passwordPrompt}"
             loginAction="#{user.login}"
             loginButtonText="#{msgs.loginButtonText}"
             managedBean="#{user}">

    <f:actionListener for="loginButton"
                     type="com.clarity.LoginActionListener"/>

  </util:login>
  ...
</html>
```

リスト 15 では、名前やパスワードのプロンプトなどの `login` コンポーネントの属性をパラメータ化しているだけでなく、アクション・リスナーをコンポーネントの Log In ボタンに接続しています。このボタンは、`login` コンポーネントのインターフェースによって公開されます (リスト 16 を参照)。

### リスト 16. `login` 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:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:composite="http://java.sun.com/jsf/composite">

  <!-- INTERFACE -->
  <composite:interface>
    <composite:actionSource name="loginButton" targets="form:loginButton"/>
    <composite:attribute name="loginButtonText" default="Log In" required="true"/>
    <composite:attribute name="loginPrompt"/>
    <composite:attribute name="namePrompt"/>
    <composite:attribute name="passwordPrompt"/>
    <composite:attribute name="loginAction"
      method-signature="java.lang.String action()"/>
    <composite:attribute name="managedBean"/>
  </composite:interface>

  <!-- IMPLEMENTATION -->
  <composite:implementation>
    <h:form id="form" prependId="false">

      <div class="prompt">
        #{cc.attrs.loginPrompt}
      </div>

      <panelGrid columns="2">
        #{cc.attrs.namePrompt}
        <h:inputText id="name" value="#{cc.attrs.managedBean.name}"/>

        #{cc.attrs.passwordPrompt}
        <h:inputSecret id="password" value="#{cc.attrs.managedBean.password}" />
      </panelGrid>

      <p>
        <h:commandButton id="loginButton"
          value="#{cc.attrs.loginButtonText}"
          action="#{cc.attrs.loginAction}"/>
      </p>
    </h:form>

    <div class="error" style="padding-top:10px;">
      <h:messages layout="table"/>
    </div>
  </composite:implementation>
</html>
```

login コンポーネントのインターフェースでは、Log In ボタンを `loginButton` という名前で公開します。この名前は `form` という名前のフォームにある Log In ボタンをターゲットとするため、`targets` 属性の値は `form:loginButton` となります。

リスト 16 の Log In ボタンに関連付けられたアクション・リスナーは、リスト 17 のとおりです。

## リスト 17. Log In ボタンのアクション・リスナー

```
package com.clarity;

import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;

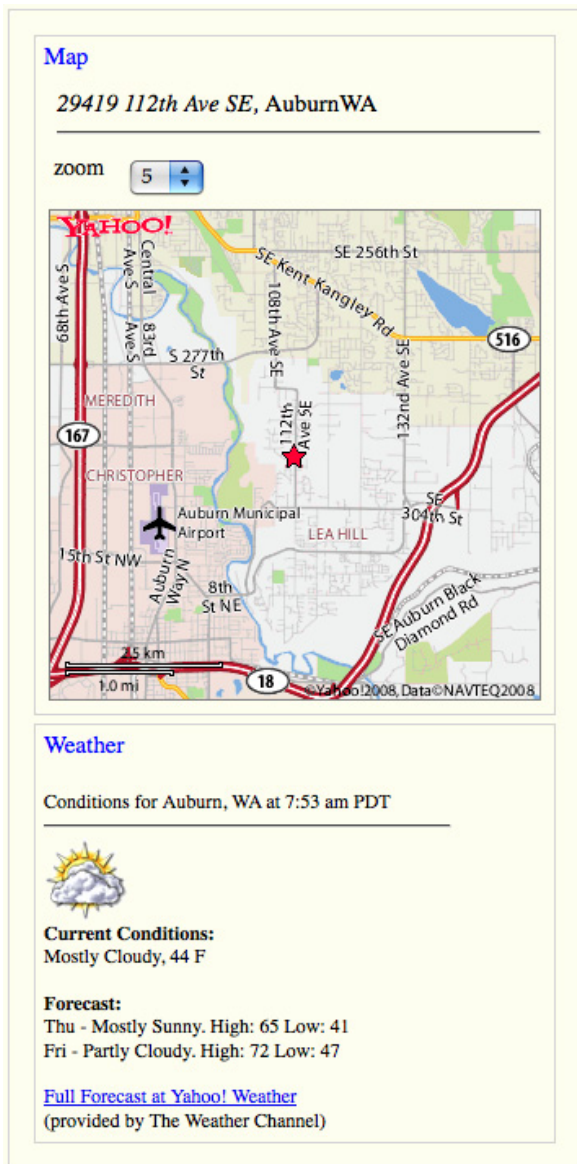
public class LoginActionListener implements ActionListener {
    public void processAction(ActionEvent e)
        throws AbortProcessingException {
        System.out.println("logging in .....");
    }
}
```

**リスト 17** のアクション・リスナーは説明用のものなので、ユーザーがログインすると単にメッセージをサーブレット・コンテナのログ・ファイルに書き込むようにしているに過ぎませんが、概念はわかってもらえるはずです。JSF 2 では完全に構成可能なコンポーネントを実装するにも、これらのコンポーネントに機能を接続するにも、Java コードや XML 構成は一切必要ありません。これはまさに強力な魅力です。

### **place** コンポーネント: 複合コンポーネントのネスト

JSF 2 では完全に構成可能なコンポーネントを Java コードや構成なしで実装できるだけでなく、複合コンポーネントをネストすることも可能です。そのため、複合コンポーネントをより管理しやすいように小さく分割することができます。例えば図 8 に示す **place** コンポーネントは、指定された住所の地図とその地域の天気の情報を表示します。

## 図 8. Places アプリケーションの **place** コンポーネント



リスト 18 に、Places アプリケーションでの **place** コンポーネントの使用方法を示します。

### リスト 18. **place** コンポーネントの使用

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:places="http://java.sun.com/jsf/composite/components/places">

  <h:form id="form">
    <ui:repeat value="#{places.placesList}" var="place">
      <places:place location="#{place}" />
    </ui:repeat>
  </h:form>
</ui:composition>
```

**place** コンポーネントのコードをリスト 19 に記載します。

## リスト 19. **place** 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:composite="http://java.sun.com/jsf/composite"
      xmlns:places="http://java.sun.com/jsf/composite/components/places">

  <!-- INTERFACE -->
  <composite:interface>
    <composite:attribute name="location" required="true"/>
  </composite:interface>

  <!-- IMPLEMENTATION -->
  <composite:implementation>
    <div class="placeHeading">

      <places:map title="Map"/>
      <places:weather title="Weather"/>

    </div>
  </composite:implementation>

</html>
```

リスト 19 の **place** コンポーネントは 2 つのネストされたコンポーネント `<places:map>` と `<places:weather>` を使用しています。リスト 20 に、`map` コンポーネントを記載します。

## リスト 20. **map** 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:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:composite="http://java.sun.com/jsf/composite">

  <!-- INTERFACE -->
  <composite:interface>
    <composite:attribute name="title"/>
  </composite:interface>

  <!-- IMPLEMENTATION -->
  <composite:implementation>
    <div class="map">
      <div style="padding-bottom: 10px;">
        <h:outputText value="#{cc.attrs.title}"
                      style="color: blue"/>
      </div>

      <h:panelGrid columns="1">
        <h:panelGroup>
          <div style="padding-left: 5px;">
            <i>
              <h:outputText value="#{cc.parent.attrs.location.streetAddress}, "/>
            </i>

            <h:outputText value=" #{cc.parent.attrs.location.city}" />
            <h:outputText value="#{cc.parent.attrs.location.state}"/><hr/>
          </div>
        </h:panelGroup>
      </h:panelGrid>
    </div>
  </composite:implementation>

</html>
```



```

<h:panelGrid columns="2">
  <div style="padding-right: 10px;margin-bottom: 10px;font-size:14px">
    #{msgs.zoomPrompt}
  </div>

  <h:selectOneMenu onChange="submit()"
    value="#{cc.parent.attrs.location.zoomIndex}"
    valueChangeListener="#{cc.parent.attrs.location.zoomChanged}"
    style="font-size:13px;font-family:Palatino">

    <f:selectItems value="#{cc.parent.attrs.location.zoomLevelItems}"/>

  </h:selectOneMenu>
</h:panelGrid>

<h:graphicImage url="#{cc.parent.attrs.location.mapUrl}"
  style="border: thin solid gray"/>

</h:panelGrid>
</div>
</composite:implementation>
</html>

```

## 複合コンポーネントのリファクタリング

リスト 20 に記載された map コンポーネントのマークアップは、私にとっては少々長過ぎる感があります。一目見ただけでは理解しづらく、その複雑さが原因で後から問題が起こることも考えられます。

リスト 20 を複数の管理しやすいファイルにリファクタリングするのは簡単です。前にリスト 8、9、10 でブレース・ビューの左側メニューについて行ったようにリファクタリングすればよいわけですが、今回このリファクタリング作業は読者の皆さんの演習として残しておきます。

リスト 20 で使用している `#{cc.parent.attrs.location.ATTRIBUTE_NAME}` という式に注目してください。複合コンポーネントの `parent` 属性を使えば親コンポーネントの属性にアクセスできるため、コンポーネントをネストするのが非常に楽になります。

その一方、ネストされたコンポーネントでは `parent` 属性だけに依存しなければならないわけでもありません。リスト 19 の `place` コンポーネントで行ったように、地図のタイトルなどの属性は親コンポーネントからそのネストされたコンポーネントに渡すことができます。これは、ネストされているかどうかに関わらず、他のあらゆるコンポーネントに属性を渡すのとまったく同じように行うことができます。

多少期待外れの感がありますが、リスト 21 は `weather` コンポーネントです。

## リスト 21. `weather` 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:h="http://java.sun.com/jsf/html"
  xmlns:composite="http://java.sun.com/jsf/composite">

  <!-- INTERFACE -->
  <composite:interface>
    <composite:attribute name="title"/>

```

```
</composite:interface>

<!-- IMPLEMENTATION -->
<composite:implementation>

  <div class="weather">
    <div style="padding-bottom: 10px;">
      <h:outputText value="#{cc.attrs.title}"
        style="color: blue"/>
    </div>

    <div style="margin-top: 10px;width:250px;">
      <h:outputText style="font-size: 12px;"
        value="#{cc.parent.attrs.location.weather}"
        escape="false"/>
    </div>
  </div>

</composite:implementation>
</html>
```

`weather` コンポーネントは `map` コンポーネントと同じように、親コンポーネントの属性 (天気情報 Web サービスからの天気に関する HTML) とコンポーネント固有の属性 (タイトル) の両方を使用します (このアプリケーションが特定の場所に対応する地図と天気の情報を取得する方法については、[第 1 回](#)を参照してください)。

つまり、ネストされたコンポーネントを実装するときには、ネストされたコンポーネントがその親コンポーネントの属性に依存するように実装することも、親コンポーネントがネストされたコンポーネントに明示的に属性を渡さなければならないように実装することもできます。例えばリスト 19 の `place` コンポーネントは、タイトルの属性をネストされたコンポーネントに明示的に渡しますが、ネストされたコンポーネントは親の属性 (地図の URL と天気に関する HTML など) に依存します。

コンポーネントの明示的な属性を実装するか、あるいは親の属性に依存するかは、結合するか、利便性を取るかのトレードオフです。この例の場合、`map` コンポーネントと `weather` コンポーネントはその親コンポーネント (`place` コンポーネント) の属性に依存するため、親コンポーネントに密接に結合されます。`map` コンポーネントと `weather` コンポーネントを `place` コンポーネントから切り離すとしたら、`map` および `weather` コンポーネントの属性すべてをコンポーネント固有の属性として指定すればよいわけですが、そうすると利便性が損なわれます。この場合、`place` コンポーネントがすべての属性を明示的に `map` および `weather` コンポーネントに渡さなければならないためです。

## 次回の予告

今回の記事では、JSF 2 のテンプレート機能と複合コンポーネントによって、簡単に保守や拡張ができる UI を実装する方法を説明しました。連載最終回となる次回の記事では、複合コンポーネントで JavaScript を使用する方法、JSF 2 の新しいイベント・モデルを適用する方法、そして JSF 2 の組み込み Ajax サポートを利用する方法を紹介します。

## ダウンロード

内容	ファイル名	サイズ
Source code for the examples	<a href="#">jsf2fu2.zip</a>	7.4MB

## 著者について

David Geary



著者、講演者、コンサルタントとして活躍する David Geary は、[Clarity Training, Inc.](#) の社長です。彼はこの会社で、開発者に JSF および GWT (Google Web Toolkit) を使用した Web アプリケーションの実装を指導しています。JSTL 1.0 および JSF 1.0/2.0 Expert Group のメンバー、そして Sun の Web 開発者認定試験の共同制作者としての経験を持つ彼は、Apache Struts や Apache Shale などのオープンソース・プロジェクトにも貢献しています。彼の著書『Graphic Java Swing』は Java 関連の本のなかでは史上に残るベスト・セラーの 1 つで、『Core JSF』(Cay Horstman との共著) は JSF 関連のベストセラー本となっています。コンファレンスやユーザー・グループで定期的に講演を行っている他、2003 年以来 NFJS ツアーの常連で、Java University では講座を受け持っています。彼は JavaOne ロック・スターに 2 回選ばれました。

© Copyright IBM Corporation 2009

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

商標

([www.ibm.com/developerworks/jp/ibm/trademarks/](http://www.ibm.com/developerworks/jp/ibm/trademarks/))