JSF 2 の魅力: 第 1 回 Web アプリケーション開発を効率化する

JSF 2 によるナビゲーションの単純化、XML 構成の排除、そしてリソースへの容易なアクセス

David Geary
President
Clarity Training, Inc.

2009年 5月 12日

JSF (Java™Server Faces) のバージョン 2.0 は、Ajax 化した堅牢な Web アプリケーションを容易に実装できるようにします。この記事は、JSF 2.0 Expert Group のメンバー、David Geary が JSF 2 の新機能を利用する方法を紹介する 3 回連載の第 1 回目です。今回の記事では、XML 構成をアノテーションおよび規約に置き換え、ナビゲーションを簡易化し、そしてリソースに簡単にアクセスできるようにすることによって、JSF 2 で開発を効率化する方法を学んでください。また、JSF アプリケーションで Groovy を使用する方法についても説明します。

このシリーズの他の記事を見る

Web アプリケーション・フレームワークが誕生する最良の場所は、知識人たちが現実とは離れて知恵を出し合う象牙の塔なのか、あるいは止むにやまれぬ必要からフレームワークが生まれてくる現実の世界なのかについては議論が続いています。直観的には、止むにやまれぬ必要のほうが知識人の集まりよりも勝っているように思えますが、この直観については掘り下げて検討する価値があるのではないでしょうか。

JSF 1 は象牙の塔で開発されましたが、それほど素晴らしい成果を上げることはなかったと言えるでしょう。しかし JSF によって市場に多くの現実的な革新技術が誕生したことは確かです。早い段階で、JSP (JavaServer Pages) に代わる強力な技術として Facelets が登場しました。それに続き、気の利いた JSF Ajax ライブラリーである Rich Faces、JSF を使用した革新的な Ajax 手法の ICEFaces、そして Seam、Spring Faces、Woodstock コンポーネント、JSF Templating などが次々と誕生しています。いずれもオープンソースの JSF プロジェクトであり、自分たちが実装する機能をフレームワークとして必要とする開発者たちによって作成されたものです。

JSF 2.0 Expert Group では基本的に、これらのオープンソース・プロジェクトによる優れた機能のいくつかを標準化しています。JSF 2 の仕様を決めたのは知識人の集団ではあるものの、その決定には現実世界の革新技術が影響していたということです。思い返せば、エキスパート・グループ

の仕事は比較的容易なものでした。私たちは Gavin King (Seam) や Alexandr Smirnov (Rich Faces)、そして Ted Goddard (ICEFaces)、Ken Paulson (JSF Templating) などの大物たちの力を借りていたからです。実際、その全員が JSF 2.0 Expert Group に参加していました。つまり、さまざまな点から考えて、JSF 2 は象牙の塔と現実世界の最良の側面を取った組み合わせと言えます。そして、JSF 2 が JSF 1 より遙かに改善されていることは確かです。

今回が 1 回目となるこの 3 回連載の記事には 2 つの目標があります。1 つは画期的な JSF 2 の機能を紹介すること、そしてもう 1 つの目標は、JSF 2 の機能を最大限に利用する方法を説明して、そのメリットを皆さんが享受できるようにすることです。この 2 つの目標を達成する手段として、JSF を最大限に利用するためのヒントと併せて JSF 2 の使用方法を説明します。今回の記事で紹介するヒントは以下のとおりです。

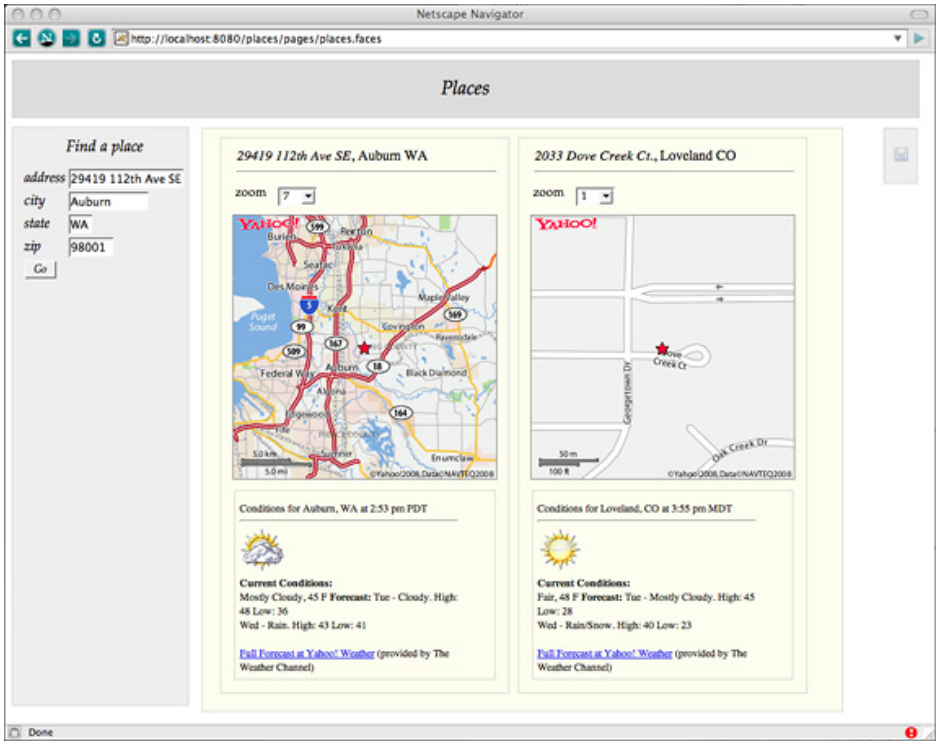
- ヒント 1: XML 構成を排除する
- ヒント 2: ナビゲーションを単純化する
- ヒント 3: Groovy を使用する
- ヒント 4: リソース・ハンドラーを使用する

以上がこの記事で取り上げるヒントですが、本題に入る前に、まずはこの連載を通して使用するサンプル・アプリケーションを紹介します。この記事のアプリケーションのサンプル・コードはダウンロードすることができます。

地図を利用した Web サービスによる、よくあるマッシュアップ・アプリケーションの例

図 1 に JSF マッシュアップ、名付けて Places アプリケーションを示します。このアプリケーションでは Yahoo! の Web サービスを使用して入力された住所情報をもとにその場所の地図を表示します。地図にはズーム機能があり、地図のほかにもその地域の天気 (予報を含む) が表示されます。

図 1. Yahoo! の Web サービスによる地図と天気の表示

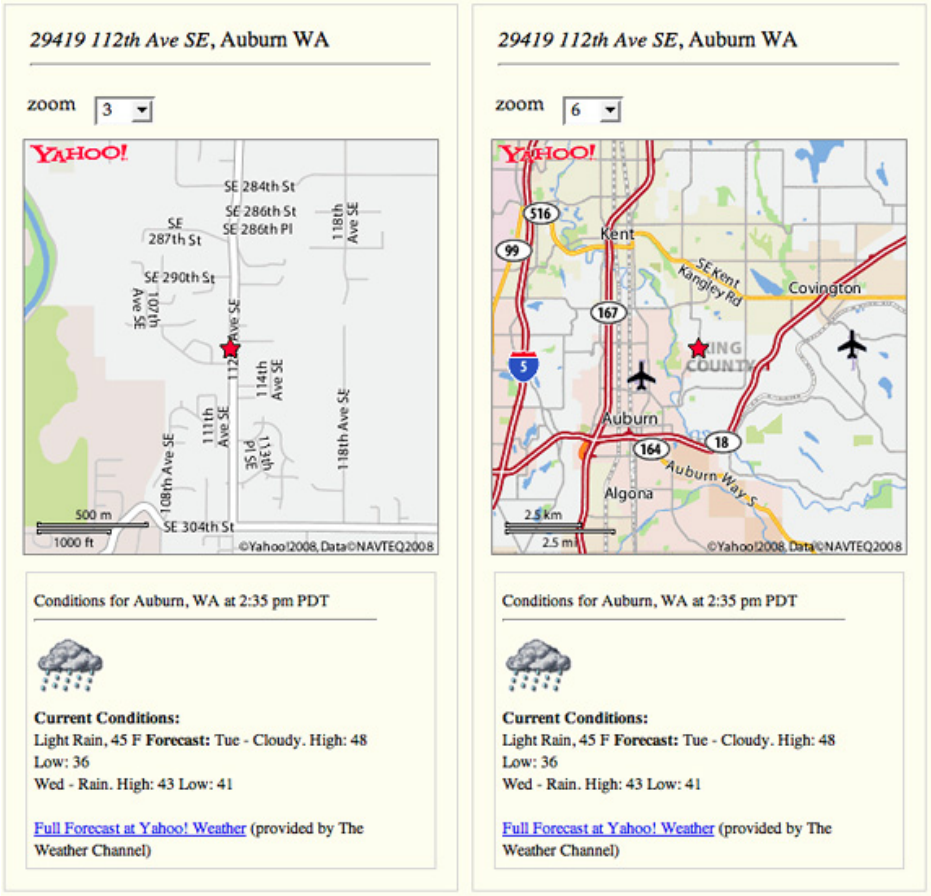


地図と天気を表示するには、住所フォームに情報を入力し、Go ボタンをクリックします。するとアプリケーションが住所を 2 つの Web サービス、Yahoo! 地図と Yahoo! 天気情報に渡します。

Yahoo! 地図のサービスが返すのは、指定された住所の地図を指す 11 個の URL で、Yahoo! サーバー上にあるこれらの地図は、それぞれにズーム・レベルが異なります。Yahoo! 天気情報サービスが返すのは、事前にアセンブルされた HTML です。画像の URL と HTML は、それぞれ `<h:graphicImage>`、`<h:outputText>` のおかげで JSF ビューに簡単に表示することができます。

Places アプリケーションでは、ユーザーが入力できる住所の数に制限はありません。図 2 に示されているように、同じ住所を複数、指定することもできます。この図は実際には、ズーム・レベルを示すために記載しています。

図 2. ズーム・レベル



アプリケーションの内容

Places アプリケーションには、表 1 に記載する 4 つの管理対象 Bean があります。

表 1. Places アプリケーションの管理対象 Bean

管理対象 Bean の名前	クラス	スコープ
`mapService`	`com.clarity.MapService`	アプリケーション
`weatherService`	`com.clarity.WeatherService`	アプリケーション
`places`	`com.clarity.Places`	セッション
`place`	`com.clarity.Place`	リクエスト

Places アプリケーションを実行するには

Places アプリケーションを実行するには、developer.yahoo.co.jp にアクセスして Yahoo! からアプリケーション ID を取得する必要があります。この ID がないと、Yahoo! の Web サービスを利用できないからです。ID を取得するには Yahoo! デベロッパーネットワークで「**アプリケーションIDの登録**」ボタンをクリックします。ID を取得したら、この記事に付属の `MapService.java` および `WeatherService.java` の中で `YOUR_ID_HERE` を自分の ID に置き換えてください。

サンプル・アプリケーションは、図 1 に示されている地図と天気の一覧 (`Places`) をセッション・スコープに保存し、その中の 1 つ (`Place`) をリクエスト・スコープで保持します。また、このア

プリケーションは、スコープをアプリケーションに設定した `mapService` 管理対象 Bean および `weatherService` 管理対象 Bean によって、Yahoo! 地図および Yahoo! 天気情報の Web サービスにそれぞれ単純な API を組み込みます。

地図と天気を表示するのは簡単です。リスト 1 に、図 1 のビューに表示された住所フォームのコードを記載します。

リスト 1. 住所フォーム

```
<h:form>
  <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="2"/>
    #{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>
```

ユーザーが Go ボタンをクリックしてフォームをサブミットすると、JSF がこのボタンのアクション・メソッド、`place.fetch()` を呼び出してWeb サービスからの情報を `Place.addPlace()` に送信します。`Place.addPlace()` は新規 `Place` インスタンスを作成し、渡されたデータでこのインスタンスを初期化してからリクエスト・スコープに保存します。

リスト 2 に、`Place.fetch()` を記載します。

リスト 2. `Place.fetch()` メソッド

```
public class Place {
  ...
  private String[] mapUrls
  private String weather
  ...
  public String fetch() {
    FacesContext fc = FacesContext.getCurrentInstance()
    ELResolver elResolver = fc.getApplication().getELResolver()

    // Get maps

    MapService ms = elResolver.getValue(
      fc.getELContext(), null, "mapService")

    mapUrls = ms.getMap(streetAddress, city, state)

    // Get weather

    WeatherService ws = elResolver.getValue(
      fc.getELContext(), null, "weatherService")

    weather = ws.getWeatherForZip(zip, true)

    // Get places

    Places places = elResolver.getValue(
      fc.getELContext(), null, "places")
```

```
    // Add new place to places

    places.addPlace(streetAddress, city, state, mapUrls, weather)

    return null
  }
}
```

`Place.fetch()` は JSF の変数リゾルバーを使用して `mapService` および `weatherService` 管理対象 Bean を検出し、この 2 つの管理対象 Bean を使用して Yahoo! の Web サービスから地図と天気の情報を取得します。続いて `fetch()` が呼び出す `places.addPlace()` は、取得された地図と天気の情報、そして住所を使用してリクエスト・スコープに新しい `Place` オブジェクトを作成します。

`fetch()` は `null` を返すことに注意してください。`fetch()` はボタンに関連付けられたアクション・メソッドです。このメソッドが `null` 値を返すことによって JSF が同じビューをリロードし、ユーザーのセッションですべての住所の地図と天気を表示することになります (リスト 3 を参照)。

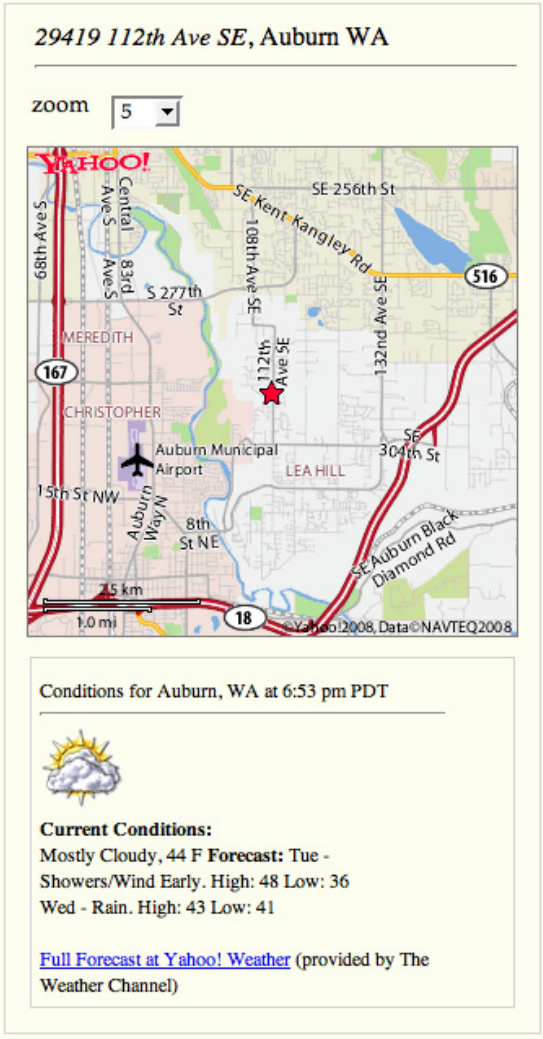
リスト 3. ビューでの地図と天気の表示

```
<ui:composition 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">

  <h:form>
    <!-- Iterate over the list of places -->
    <ui:repeat value="#{places.placesList}" var="place">

      <div class="placeHeading">
        <h:panelGrid columns="1">

          <!-- Address at the top -->
          <h:panelGroup>
            <div style="padding-left: 5px;">
              <i><h:outputText value="#{place.streetAddress}"/></i>,
              <h:outputText value="#{place.city}"/>
              <h:outputText value="#{place.state}"/>
              <hr/>
            </div>
          </h:panelGroup>

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

            <!-- dropdown -->
            <h:selectOneMenu onchange="submit()"
                 value="#{place.zoomIndex}"
                 valueChangeListener="#{place.zoomChanged}"
                 style="font-size:13px;font-family:Palatino">

              <f:selectItems value="#{places.zoomLevelItems}"/>

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

          <!-- The map -->
          <h:graphicImage url="#{place.mapUrl}" style="border: thin solid gray"/>
```

```
        </h:panelGrid>

        <!-- The weather -->
        <div class="placeMap">
          <div style="margin-top: 10px;width:250px;">
            <h:outputText style="font-size: 12px;"
              value="#{place.weather}"
              escape="false"/>
          </div>
        </div>
      </div>

    </ui:repeat>
  </h:form>

</ui:composition>
```

リスト 3 のコードは Facelets の <ui:repeat> タグを使用して、ユーザーのセッションに保存された住所のリストをループ処理します。それぞれの住所ごとに、図 3 のような出力が表示されます。

図 3. ビューに表示された地図と天気



ズーム・レベルの変更

ズーム・メニュー (図 3 およびリスト 3 を参照) には `onchange="submit()"` 属性が設定されています。そのため、ユーザーがズーム・レベルを選択すると JavaScript の `submit()` 関数はメニューの周辺フォームをサブミットします。このフォームがサブミットされることによって JSF が呼び出すのが、メニューに関連付けられた値変更リスナー、`Place.zoomChanged()` メソッドです (リスト 4 を参照)。

リスト 4. `Place.zoomChanged()`

```
public void zoomChanged(ValueChangeEvent e) {
  String value = e.getComponent().getValue()
  zoomIndex = (new Integer(value)).intValue()
}
```

`Place.zoomChanged()` は、`Place` クラスのメンバー変数 `zoomIndex` にズーム・レベルを保存します。サーバーに対するこの送信によってナビゲーションが影響されることはないため、JSF は同じページをリロードし、マップを新しいズーム・レベル (`<h:graphicImage url="#{place.mapUrl}..."/>` など) で更新します。画像が描画されると、JSF は `Place.getMapUrl()` を呼び出し、このメソッドが現行のズーム・レベルに対応する地図の URL を返します (リスト 5 を参照)。

リスト 5. `Place.getMapUrl()`

```
public String getMapUrl() {
  return mapUrls == null ? "" : mapUrls[zoomIndex]
}
```

Facelets の使用

JSF 1 を使用した経験がある方は、この記事に記載した JSF 2 のコードには JSF 1 との微妙な違いがあることにお気付きかと思います。まず、ここで使用しているのは JSP ではなく、JSF 2 の新しい表示技術、Facelets です。連載の今後の記事で説明しますが、Facelets は堅牢かつ柔軟で拡張可能なユーザー・インターフェースを実装するための強力な機能を数多く提供します。しかし、これまでに記載したコード・リストでは、その威力をそれほど利用していません。Facelets が JSF にもたらす多くのマイナーな改善内容の 1 つは、JSF 値の式を直接 XHTML ページに組み込めるようにすることです。例えばリスト 1 では、`#{msgs.city}` などの式をページに直接組み込んでいます。JSF 1 の場合、この式は `<h:outputText>` でラップして `<h:outputText value="#{msgs.city}"/>` のようにしなければなりませんでした。ただし注意する点として、セキュリティー上の理由から、ユーザーが入力したテキストは常にエスケープする必要があります。そのため、リスト 3 では住所情報を表示するために `<h:outputText>` を使用してデフォルトでそのテキストをエスケープしています。

Facelets の観点からもう 1 つ注意しなければならないのは、リスト 3 の `<ui:composition>` タグです。このタグは、リスト 3 の XHTML ページが他の XHTML ページに組み込まれることを意味します。Facelets の構成は、Facelets テンプレートを作成する際の中心的要素で、このテンプレートはよく使われている Apache Tiles フレームワークに似ています。連載の以降の記事では Facelets テンプレートを取り上げ、Smalltalk パターンの Composed Method パターンに従ってビューを構成する方法を説明します。

Facelets を使用するという以外、これまでのコードに JSF 1 との大きな違いはありませんでしたが、ここからは、それよりも顕著な違いについて説明します。第一の違いは、JSF 2 アプリケーションの場合に作成する XML 構成の量です。

ヒント 1: XML 構成を排除する

Web アプリケーションを XML で構成することについては、これまで常に疑問がありました。XML 構成は厄介で間違いの元となるため、例えばアノテーション、規約、あるいはドメイン固有言語などによりフレームワークに任せるのが最善です。開発者としては、冗長な XML で細部を沢山作成する代わりに、仕事を完成させることに専念できるようでなければなりません。

この点を明らかにする例として、リスト 6 に、JSF 1 で Places アプリケーションの管理対象 Bean を宣言するために必要な 20 行の XML を記載します。

リスト 6. JSF 1 での管理対象 Bean の宣言

```
<managed-bean>
  <managed-bean-class>com.clarity.MapService</managed-bean-class>
  <managed-bean-name>mapService</managed-bean-name>
  <managed-bean-scope>application</managed-bean-scope>
</managed-bean>

<managed-bean>
  <managed-bean-class>com.clarity.WeatherService</managed-bean-class>
  <managed-bean-name>weatherService</managed-bean-name>
  <managed-bean-scope>application</managed-bean-scope>
</managed-bean>

<managed-bean>
  <managed-bean-class>com.clarity.Places</managed-bean-class>
  <managed-bean-name>places</managed-bean-name>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

<managed-bean>
  <managed-bean-class>com.clarity.Place</managed-bean-class>
  <managed-bean-name>place</managed-bean-name>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

JSF 2 の場合、リスト 7 に示すとおり XML は消えてなくなっています。JSF 2 では代わりに、クラスにアノテーションを付けることになります。

リスト 7. JSF 2 での管理対象 Bean のアノテーション

```
@ManagedBean(eager=true)
public class MapService {
  ...
}

@ManagedBean(eager=true)
public class WeatherService {
  ...
}

@ManagedBean()
@SessionScoped
public class Places {
  ...
}
```

```
@ManagedBean()
@RequestScoped
public class Place {
  ...
}
```

規約では、管理対象 Bean の名前はクラス名と同じで、クラス名の先頭文字が小文字に変換されます。リスト 7 で作成した管理対象 Bean の場合で言うと、クラス名は上から順に `mapService`、`weatherService`、`places`、`place` となります。さらに、`ManagedBean` アノテーションの `name` 属性を使って管理対象 Bean の名前を明示的に指定することもできます (例えば、`@ManagedBean(name = "place")`)。

リスト 7 の `mapService` および `webService` 管理対象 Bean には `eager` 属性を使用しました。`eager` 属性が `true` であれば、JSF は起動時に管理対象 Bean を作成してアプリケーション・スコープに配置します。

`@ManagedProperty` アノテーションを使って管理対象 Bean のプロパティーを設定することもできます。表 2 に、すべての JSF 2 管理対象 Bean アノテーションを示します。

表 2. JSF 2 管理対象 Bean アノテーション (`@...Scoped` アノテーションは `@ManagedBean` と併用した場合にのみ有効)

管理対象 Bean アノテーション	説明	属性
`@ManagedBean`	このクラスのインスタンスを管理対象 Bean として登録し、`@...Scoped` アノテーションのいずれかで指定されたスコープに配置します。スコープが指定されていない場合、JSF は Bean をリクエスト・スコープに配置します。名前が指定されていなければ、JSF がクラス名の先頭文字を小文字に変換して管理対象 Bean 名にします。例えば、クラス名が `UserBean` だとすると、JSF は `userBean` という名前の管理対象 Bean を作成します。`eager` 属性と `name` 属性はいずれもオプションです。 引数を使用しないコンストラクターを実装する Java クラスには、このアノテーションを使用する必要があります。	`eager`, `name`
`@ManagedProperty`	管理対象 Bean のプロパティーを設定します。このアノテーションは、クラスのメンバー変数宣言の前に配置する必要があります。プロパティーの名前は、`name` 属性が指定します。デフォルトではプロパティー名がメンバー変数の名前に設定されます。`value` 属性はプロパティーの値で、文字列または JSF 式 (`#{...}` など) のいずれかになります。	`value`, `name`
`@ApplicationScoped`	管理対象 Bean をアプリケーション・スコープに保存します。	
`@SessionScoped`	管理対象 Bean をセッション・スコープに保存します。	
`@RequestScoped`	管理対象 Bean をリクエスト・スコープに保存します。	
`@ViewScoped`	管理対象 Bean をビュー・スコープに保存します。	

`@NoneScoped`	管理対象 Bean にスコープがないことを指定します。スコープのない管理対象 Bean は、他の Bean で参照するときに便利です。	
`@CustomScoped`	管理対象 Bean をカスタム・スコープに保存します。 カスタム・スコープとは、単にページ作成者がアクセスできるマップのことです。カスタム・スコープでは、Bean の可視性と存続期間をプログラムによって制御することができます。マップを指定するのは `value` 属性です。	`value`

faces-config.xml から管理対象 Bean 宣言を削除すれば、XML の量が大幅に減ります。一方で JSF 2 では、この記事の管理対象 Bean の場合と同じようにアノテーションを使用するか、または JSF 2 の単純化されたナビゲーション操作の場合のように規約を使用することによって、事実上、XML を完全に排除することができます。

ヒント 2: ナビゲーションを単純化する

JSF 1では、XML でナビゲーションを指定していました。例えば login.xhtml から places.xhtml に移動する場合には、リスト 8 のナビゲーション・ルールを使用することになります。

リスト 8. JSF 1 でのナビゲーション構成ルールおよびケース

```
<navigation-rule>
  <navigation-case>
    <from-view-id>/pages/login.xhtml</from-view-id>
    <outcome>places</outcome>
    <to-view-id>/pages/places.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

リスト 8 の XML を排除するには、JSF 2 のナビゲーション規約を利用することができます。つまり、JSF がボタンのアクションの最後に .xhtml を追加して、このファイルをロードするというものです。これは、規約さえあれば、アノテーションやその他何もなくてもナビゲーション・ルールの作成作業を完全になくすことができることを意味します。リスト 9 ではボタンのアクションが `places` なので、JSF は places.xhtml をロードします。

リスト 9. 規約によるナビゲーション

```
<h:commandButton id="loginButton"
  value="#{msgs.loginButtonText}"
  action="places"/>
```

リスト 9 にナビゲーション XML は一切必要ありません。リスト 9 のボタンは places.xhtml をロードしますが、それは、ボタンが置かれているファイルと同じディレクトリーにこのファイルが置かれている場合のみです。アクションがスラッシュ (`/`) で始まっていなければ、JSF はそのパスを相対パスであると想定します。これよりも明示的なパスにするには、リスト 10 のように絶対パスを指定することができます。

リスト 10. 絶対パスを使用したナビゲーション

```
<h:commandButton id="loginButton"
  value="#{msgs.loginButtonText}"
  action="/pages/places"/>
```

ユーザーがリスト 10 のボタンをクリックすると、JSF は /pages/places.xhtml ファイルをロードします。

デフォルトでは、JSF は XHTML ページから次の XHTML ページに進みますが、`faces-redirect` パラメーターを指定してリダイレクトすることも可能です (リスト 11 を参照)。

リスト 11. リダイレクトによるナビゲーション

```
<h:commandButton id="loginButton"
  value="#{msgs.loginButtonText}"
  action="places?faces-redirect=true"/>
```

ヒント 3: Groovy を使用する

Java 技術の最大の利点は Java 言語ではなく、Java 仮想マシン (JVM) です。Scala、JRuby、Groovy といった強力で革新的な新しい言語は JVM 上で動作し、コードの作成に別の側面をもたらします。これらの革新的言語のなかでも特によく使われているのが、Groovy です。名前は不適切ながらも極めて機能性に優れたこの言語には、Ruby、Smalltalk、そして Java 言語が一体化されています (「参考文献」を参照)。

Groovy を使用する理由はさまざまにありますが、まずは、Groovy はそのまたいとこである Java 言語より遙かに簡潔で、しかも遙かに強力であることです。さらに 2 つの理由を挙げるとしたら、Groovy にはセミコロンもキャストも必要ありません。

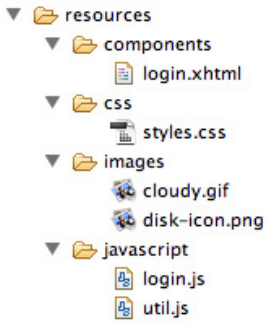
お気付きではないかもしれませんが、リスト 2 の `Place` クラスは Groovy で作成されています。セミコロンがないことが多少の手掛かりとなる一方、注目してもらいたいのは `MapService ms = elResolver.getValue(...)` という行です。`elResolver.getValue()` メソッドは `Object` 型を返すことから、Java コードの場合にはメソッドの実行結果をキャストしなければなりませんが、Groovy はそのキャストを代わりに行ってくれます。

Groovy は通常 Java コードで作成するような JSF 成果物 (例えばコンポーネント、レンダラー、バリデーター、コンバーターなど) のどれにでも使用することができます。実のところ、これは JSF 2 で始まったことではありません。Groovy ソース・ファイルは Java バイトコードにコンパイルされるため、Groovy が生成した .class ファイルは、`javac` によって生成されたかのようにそのまま使用することができます。もちろん、コードが機能するようになると同時に Groovy ソース・コードをホット・デプロイする方法を知る必要が出てくるはずですが、Eclipse のユーザーにとってその答えは単純です。つまり、Groovy Eclipse プラグイン (「参考文献」を参照) をダウンロードしてインストールすればよいだけのことです。Sun の JSF 実装である Mojarra には、バージョン 1.2_09 から明示的な Groovy サポートが備わっています (「参考文献」を参照)。

ヒント 4: リソース・ハンドラーを使用する

JSF 2 はリソースの定義やリソースへのアクセスを行うための標準メカニズムを提供します。リソースは最上位レベルの resources ディレクトリー配下に配置し、いくつかの JSF 2 タグを使用して、これらのリソースにビュー内でアクセスします。一例として、図 4 に Places アプリケーションのリソースを示します。

図 4. Places アプリケーションのリソース



リソースの要件は唯一、それが resources ディレクトリーまたはそのサブディレクトリーに置かれていることだけです。resources ディレクトリーのサブディレクトリーには任意の名前を指定することができます。

ビュー・コードでリソースにアクセスするには、`<h:outputScript>` および `<h:outputStylesheet>` という 2 つの JSF 2 タグを使用します。これらのタグは JSF 2 の `<h:head>` タグと `<h:body>` タグと連携して機能します (リスト 12 を参照)。

リスト 12. XHTML でのリソースへのアクセス

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

  <h:head>
    ...
  </h:head>

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

`<h:outputScript>` および `<h:outputStylesheet>` タグには、それぞれスクリプト、スタイルシートを識別する `library` と `name` という 2 つの属性があります。`library` 属性の名前は resources ディレクトリーの下にあるディレクトリーに対応し、そのディレクトリーにリソースが置かれています。例えばスタイルシートが resources/css/en ディレクトリーにある場合、`library` 属性は `css/en` となります。`name` 属性は、リソースそのものの名前です。

再配置可能なリソース

ページ内でリソースを表示する位置は、開発者が指定可能であることが重要です。例えばページの body に JavaScript を配置すると、ブラウザーはそのページをロードするときに JavaScript を実行します。一方、ページの head に JavaScript を配置した場合には、その JavaScript は呼び出されたときにしか実行されません。リソースの配置場所によってその使用方法は左右されるため、リソースが最終的に配置される場所を開発者が指定できるようでなければなりません。

JSF 2 のリソースは再配置可能です。これは、リソースを配置するページ内の場所を指定できるということを意味します。場所を指定するには `target` 属性を使用します。例えばリスト 12 を見ると、CSS は body に配置され、JavaScript は head に配置されています。

場合によっては、JSF 式言語 (EL: Expression Language) を使用してリソースにアクセスしなければならないこともあります。リスト 13 に、例として `<h:graphicImage>` で画像にアクセスする方法を示します。

リスト 13. JSF 式言語 によるリソースへのアクセス

```
<h:graphicImage value="#{resource['images:cloudy.gif']}"/>
```

EL 以外でリスト 13 の構文を作成する方法

認めざるを得ないことですが、リスト 13 の構文はいくらか直観に反したものになっています。この構文は実際には、リソースを保存するために JSF によって作成されたマップにアクセスするものです。したがって、この構文を使用することは稀にしかありません。実のところ、EL を使用しなくても、`<h:graphicImage/>` を使って `<h:graphicImage library="images" name="cloudy.gif"/>` のようにすれば、画像にアクセスすることができます。

EL 式のなかでリソースにアクセスするための構文は、`resource['LIBRARY:NAME']` です。ここで、`LIBRARY` と `NAME` は `<h:outputScript>` タグと `<h:outputStylesheet>` タグの `library` および `name` 属性に対応します。

今後の予告

今回の記事では管理対象 Bean アノテーション、ナビゲーションの単純化、そしてリソースのサポートを取り上げましたが、まだ JSF 2 の機能を表面的にかじっただけに過ぎません。連載の今後の 2 回の記事では、Facelets、JSF 2 の複合コンポーネント、組み込み Ajax サポートについて探っていきます。

ダウンロード

内容	ファイル名	サイズ
Source code	jsf2fu1.zip	1.9MB

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。