- JavaServer Faces って何？
- これまでの Web アプリ技術とどう違うの？
- JSF の得意技は何？
- JSF を勉強していくのに何が必要か？

1章 JavaServer Faces をよろしく!

ここんとこJavaは、Webアプリケーションなら Java、と言われるほど、Web開発のトップ技術にのし上がってきました。たしかにサーブレットやJSPを使うと、Webブラウザをユーザインタフェイスとするいろんなアプリケーションを、スケーラビリティの点でも堅牢性（ロバストネス）の面でも安心して開発〜展開できる例が多いのです。しかし最近はWebアプリケーションの要件が複雑になり、HTMLのインタフェイスでは力不足だ、イベント駆動方式の本格的なGUIを使いたい、という声が高まってきました。サーブレットとJSPがいくら便利でも、HTTPは、ステートを持てない、あまりにも単純素朴な応答プロトコルですから、AWT や Swing のような Java の GUI でできることを Web でやろうとすると、ものすごく苦労しなければなりません。

Webアプリケーションのユーザインタフェイスを本物のGUIのように高度化しようという試みが、オープンソースと商用ソフトの両方に、これまでいくつかありました。EnhydraのBarracuda、Apache の Tapestry、Oracle の UIX、Sun の JATO などが中でもよく知られています。2001 年の春に、これら各社各組織の技術者たちが集まってJavaコミュニティプロセス（Java Community Process, JCP）のグループを作りました。Webアプリケーションのための高度なユーザインタフェイスを標準化することが、目的です。私も、最初からこのグループのメンバーです。そこから生まれた標準規格、それが JavaServer Faces です。JavaServer Faces の規格のバージョン 1.0 は 2004 年の 3 月に発表されました†。

1.1 JavaServer Faces って何？

JavaServer Faces（JSF）を使うと、複雑高度なWebアプリケーションのユーザインタフェイスの開発が楽になります。HTTPのリクエスト処理の各段階は、HTTPの規格としてすでに明確に定義されていますが、JavaServer Facesの中心的な考え方は、リクエスト処理のひとつひとつの段階に、ユーザインタフェイスコンポーネントのモデル（データ形式、イベント形式、コンポーネントクラスが持つメソッド集合など）を一対一対応で結び付けることです。それによって、次のことが可能になります：

† 訳者注：1.1 が 5 月末に発表された。

- アプリケーションのバックエンド(基本ロジック)を、HTTPをいちいち意識せずに書けます。また、アプリケーションのロジックとユーザインタフェイスを、JavaやWindowsのプログラマにはおなじみの、ふつうのイベント処理の仕組みで結び付けることができます。そのために提供されているAPIは、きわめて汎用性に富んでいます。
- Webページ(HTMLドキュメントなど)は書くがプログラミングはしない、という人びとでも、コンポーネントを組み合わせることによって、自分の好みに合ったユーザインタフェイスを構築できます。ユーザとの対話やユーザからの入力などの機能は、JSFのコンポーネントが提供します。したがって、インタフェイスの中にプログラムのロジックの一部を書かなければならない状態、つまりプログラマがページの一部を書かなければならないような状態を、極力避けることができます。
- JavaServer Facesを核として各社は、Webアプリケーションのフロントエンドやバックエンドを開発するための便利なツールを作れます。

というより、JSF自体は規格であり、実装は各ベンダや各組織(ときには個人デベロッパ)が作るものです。JSFの規格は、ユーザインタフェイスを構成するための一連のコンポーネントを定義しています。それらは、HTMLのFORMの背後に機能を付け加えたようなものです。それらのコンポーネントをそのまま使ってもいいし、またJSFの標準APIをベースにして拡張コンポーネントを書くこともできます。各コンポーネントにはヴァリデータが付いていて、ユーザの入力を検査し、検査後のデータをアプリケーションオブジェクトに自動的に送ります。ユーザがボタンやリンクをクリックするとイベントハンドラが起動し、イベントハンドラの処理によって、そのほかのコンポーネントのステートを変えたり、アプリケーション本体のコードを呼び出したりします。イベント処理の結果として次に表示するページが決まりますが、そういうページ遷移のストーリーの制御を、ナビゲーションハンドラというものが支援します。ナビゲーションハンドラ(ページ遷移のストーリーを制御する部位)は、独自の実装を差し替えて使うことができます。

HTMLはほとんどのWebアプリケーションで使われるマークアップ言語です。本書の中のコードも、ほとんどHTMLばかり使っています。しかしJSFは、マークアップ言語を特定しません。JSFには、UIのコンポーネントの名前や機能の定義とは別に、レンダラ(renderer,作画系/表示系)という差し替え可能な部位があります。レンダラとしてHTMLを指定すれば、HTMLのFORMがUIを表示するでしょう。あるいはレンダラとしてWMLを指定すれば、UIの実際の表示はWMLが行います。このような仕組みは、Swingの“プラッガブルなルック＆フィール(PLAF)”†と似ていますね。

JSFの実装はプレゼンテーションレイヤ(presentation layer,システム全体の中でプレゼンテーション(表示/描画)を担当する層)として必ずJavaServer Pages(JSP)を使わなければなりませんし、またJSPの上でカスタムタグ††を通じてJSFのコンポーネントを使います。しかしJSFのAPIそのものは、プレゼンテーション技術をJSPに特定していません。たとえば、HTMLではなくJavaを使って

† 訳者注：ユーザインタフェイスの外見デザインを自由に変えられる仕組み。
†† 本書では“カスタムアクション”という言葉を、カスタムタグと同じ意味で使います。

JSFのコンポーネントを実装することもできます…それはSwingのやり方と似ています。あるいはまた、HTMLで書かれたテンプレートの中の要素に、JSFのコンポーネントをあてはめていくこともできます。これは、Barracude/XMLCやTapestryが使っている方法です。JSF 1.0の規格としてJSPは必須ですから本書の中のコードはほとんどJSPを使っています。しかしときどきは、ほかのプレゼンテーション形式もお見せしたいと思います。

Webアプリケーションの開発経験者は、いわゆるフレームワークというものをご存じでしょう。ApacheのStrutsや（http://jakarta.apache.org/struts/）、Maveric（http://mav.sourceforge.net/）などが有名です。JSFの機能の一部（たとえばヴァリデーションやナビゲーション制御の部分）は、これらのフレームワークも持っていますが、既存のフレームワークとJSFを併用することも十分に可能です。StrutsのアプリケーションがJSFのインタフェイスを使う例を本書の１２章で説明しますから、この人気の高いフレームワークとJSFの相性の良さを、読者も確認できるでしょう。

1.2 これまでのWebアプリ技術とどう違うの？

JSFはWebアプリケーションの世界に、UIコンポーネントとその上のイベントを軸とする処理形式を持ち込みます。そういう処理形式（イベントがコールバック†を呼び出す、という処理形式）は、従来のコンピュータ、とくにパソコンの上のGUIで長年使われてきました。ではこの形式は、従来のWebアプリケーション技術と比べて、どういう点で有利なのでしょうか？。

1.2.1 最少限のコードでユーザインタフェイスを記述できる

Javaで作るWebアプリケーションのユーザインタフェイスは、主に、一連のJSPページで実装します（VelocityやFreeMakerのようなテンプレートエンジンのためのページを書くこともあります）。JSPやテンプレートのページの上では、静的なコンテンツ（テキストとHTMLの成分）と動的なコンテンツを生成する部分が入り混じっています。しかしこのようなページの問題点は、ページ全体がユーザインタフェイスを成立させるためのロジックに追われてしまいがちなことです。チェックボックスのどこにチェックを入れるか、コンボボックスのどのアイテムをselectedにするか、入力欄には最初何を表示するのか、等々の記述で、ページがうまってしまいます。

しかしJSFを使う前提でJSPを書けば、JSFのコンポーネントをカスタムタグで表現できます。たとえば以下の例は、ユーザに好物（好きな食べ物）を選ばせるチェックボックスの集まりです：

```
...
<h:form>
  <table>
  ...
    <tr>
      <td>Favorite Foods:</td>
      <td>
        <h:selectManyCheckbox value="#{cust.foodSelections}">
          <f:selectItem itemValue="z" itemLabel="Pizza" />
```

† 訳者注：ユーザ＝プログラマが登録したイベントハンドラ。

```
            <f:selectItem itemValue="p" itemLabel="Pasta" />
            <f:selectItem itemValue="c" itemLabel="Chinese" />
          </h:selectManyCheckbox>
        </td>
      </tr>
      ...
    </table>
  </h:form>
  ...
```

現段階で、このコードを詳しく理解する必要はありません。ご覧になって気づいていただきたいのは、ループや条件文のようなややこしいロジックがまったくないことです。それらのロジックは、<h:selectManyCheckbox>とか<f:selectItem>といったカスタムタグで記述するJSFのコンポーネントが内部で実装しています。フォームが送信されると、JSFはユーザが指定～入力した一連のデータをサーバ上のアプリケーションオブジェクトに保存します。そのアプリケーションオブジェクトが、上のコードでは#{cust.foodSelections}という式で指定されています。つまりアプリケーションオブジェクトを、カスタムタグのvalue属性で指定します。リスポンスの内容が表示されるときJSFは、そのユーザデータに基づいてそれぞれのチェックボックスにチェックを入れます。しかし以上のようなロジックは、ご覧のようにページの字面(じづら)にはぜんぜん現れていません。ですから、ページを書く人の仕事がずいぶん楽になります。

プレゼンテーションレイヤとしてJSPを使うときには、ページの作者がJSFのカスタムタグの書き方を覚えなければなりません。しかし彼や彼女のお気に入りのページ作成ツールは、JSFのタグをまったく理解しないでしょう。ですが、前に述べたように、JSFではプレゼンテーションレイヤとしてJSP以外のものも使えます。1.0の規格には詳しく書かれていませんが、JSFのAPIを使うJSP以外のプレゼンテーション層を一から作ることは十分に可能です。たとえば、見た目にはHTMLだけで書かれているが、バックではそれらのロジックをJSFのAPIがサポートしている、という方式もありえます。そうするとJSFの楽屋裏の部分をプログラマが書き、ページの作者は自分が使い慣れているページ作成ツール(JSFのことを知らないツール)と自分がよく知っているHTMLだけを使ってページを書いていけます：

```
...
<form action="validate_jstl.jsp" method="post">
  <table>
  ...
    <tr>
      <td>Favorite Foods:</td>
      <td>
        <input id="pizza" type="checkbox" name="food" value="z">Pizza<br>
        <input id="pasta" type="checkbox" name="food" value="p">Pasta<br>
        <input id="chinese" type="checkbox" name="food" value="c">Chinese<br>
      </td>
    </tr>
    ...
  </table>
</form>
...
```

上の例で、ふつうの書き方とやや違うのは、動的成分を表現する成分にid属性があることです。このid属性によって、このテンプレートを支える独自のプレゼンテーション層がJSFのコンポーネントをテンプレート(一見HTMLだけのテンプレート)の成分に結び付けることができます。idによってコンポーネントは、どの成分を変えるのか(チェックを入れるのか)を知ることができます。このようなやり方を使う独自のプレゼンテーション層の作例を、本書の終わりのほうで見ていただきます。

このような、チェックボックスがいくつかあるだけという簡単な例からも、これまでのJSPページにくらべてJSFはずいぶん簡単だ、とお感じになったでしょう。これまでの書き方ですと、たとえばこうなります：

```
...
<form action="validate_jstl.jsp" method="post">
  <table>
  ...
    <c:forEach items="${paramValues.food}" var="current">
      <c:choose>
        <c:when test="${current == 'z'}">
          <c:set var="pizzaSelected" value="true" />
        </c:when>
        <c:when test="${current == 'p'}">
          <c:set var="pastaSelected" value="true" />
        </c:when>
        <c:when test="${current == 'c'}">
          <c:set var="chineseSelected" value="true" />
        </c:when>
      </c:choose>
    </c:forEach>
    <tr>
      <td>Favorite Foods:</td>
      <td>
        <input type="checkbox" name="food" value="z"
          ${pizzaSelected ? 'checked' : ''}>Pizza<br>
        <input type="checkbox" name="food" value="p"
          ${pastaSelected ? 'checked' : ''}>Pasta<br>
        <input type="checkbox" name="food" value="c"
          ${chineseSelected ? 'checked' : ''}>Chinese
      </td>
    </tr>
    ...
  </table>
</form>
...
```

上のコードは、JSPと、JSPの標準タグライブラリ(JSP Standard Tag Library, JSTL)を知らない人が見ると??でしょうが、しかしコードを詳しく理解する必要はありません。ぱっと見て気づいていただきたいのは、このページはHTMLの成分に加えて、プログラム的な部分がとても多いことです(実際にはXMLふうに書いたJSPのアクションとタグを記述する式言語だけですが、かなりややこしいことは事実です)。最初のほうのループでは、foodという名前のリクエストパラメータから受け

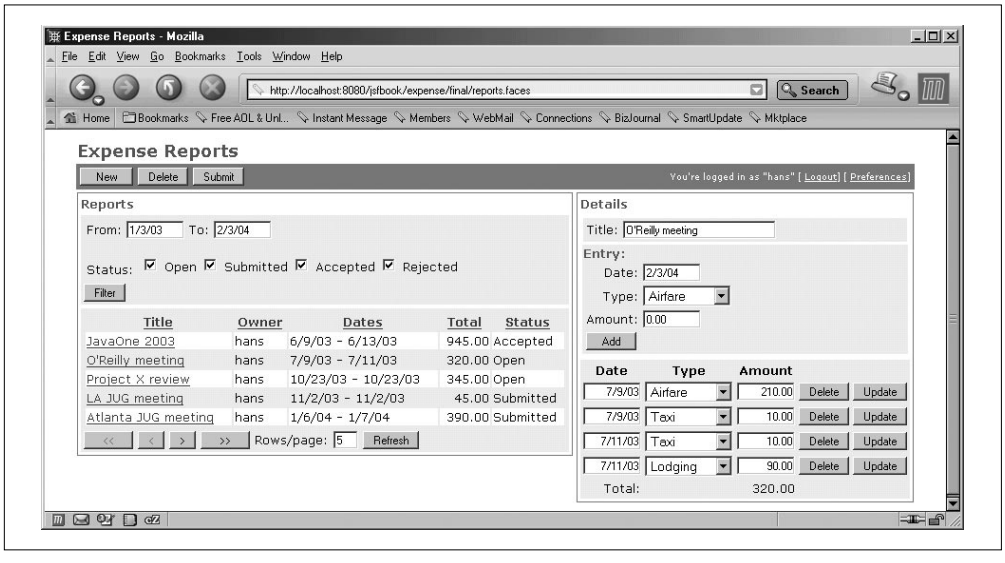

図 1-1　複雑な Web アプリケーションのユーザインタフェイスの例

取った値を調べ、それに基づいてvalueというフラグをセットしています。次にこれらのフラグを使って、チェックボックスにcheckedという属性を加えるかどうかを決めています。

このような書き方で、たとえば図1-1のような複雑なページの、動的に決まる値をすべて表示しようとすると、どれだけたくさんのコードを書かなければならないでしょうか？。ちょっと、想像してみてください。

この図1-1のページは、本書の中で作っていくサンプルアプリケーションのメイン画面です。ボタンが多数あって、それらが、誰が今ログインしているか、ビジネスオブジェクトのステータスはどうなっているか、などに応じて、イネーブル（有反応）になったりディスエーブル（無反応）になったりします。テーブルがあり、その中身はソートされ、スクロールします。たくさんある入力欄には最初、前のリクエストの値が表示されます。また、ユーザがクリックしたボタンやリンクの種類によって、それぞれ違うリクエストが生成され、アプリケーションに送られます。このような複雑なユーザインタフェイスとその背後のロジックを持つアプリケーションを作るためには、JSF が最適です。

1.2.2　ユーザインタフェイスのコードを明確にモジュール化できる

ベテランが推奨するWebアプリケーションの基本形が、いわゆるMVC（Model-View-Controller）デザインパターンです。MVC は1980年代の後期にXeroxの人たちがSmalltalkを開発したとき、いろんな論文の中で提唱されました。そしてこの形式はその後、Smalltalkにかぎらず、さまざまなプログラミング言語で書かれるGUIのアプリケーションで、広く使われるようになりました。MVCはまず、アプリケーションのデータやビジネスロジックを“モデル（Model）”として独立させます。また、データの表現（プレゼンテーション）を“ビュー（View）”としてやはり独立させます。そしてさらに、

データに対するユーザの対話的な操作を“コントローラ(Controller)”として独立させます。

モデルは、ビジネスデータやデータの使い方のルールを表現します。データがどのように表示されるのか、とか、どんなユーザインタフェイスによってデータを変えるのか、という部分にはモデルはいっさい関知しません。これに対してビューは、ユーザインタフェイスの細かい部分をすべて担当します。またビューは、モデルのデータの正しい読み方や表示の仕方も知っています。さらにまたビューは、ユーザインタフェイスの上で起きたこと…すなわちイベント…に基づいてコントローラに指令します。そしてコントローラが、モデルのデータを実際に書き換えます。

このようなMVCという形を使うと、ひとつのビジネスオブジェクトに対して複数のビューを使えますし、また他方では、モデルの側はユーザインタフェイスのことをまったく気にせずにデータやビジネスルールを変えることができます。すなわち、アプリケーション全体の構造に、安全性と柔軟性が備わります。

Strutsのような、Javaで作るWebアプリケーションのためのフレームワークも、ずっと上のレベルのオブジェクトではMVCをサポートしていますが、FORMを処理したりする下のほうの細かいレベルでは、本格的なGUIアプリケーションのようなMVCに基づくモジュール化、すなわち要素の分割と独立が徹底していません。たとえばStrutsでは、ビューを一連のJSPページで表現し、コントローラをStrutsのサーブレット(とくにActionクラス)で表現し、そしてモデルをアプリケーションクラスで表現します。大枠ではこのように、一応MVCです。モデルに該当するアプリケーションクラスは通常、データベースのような保存性のデータを操作することが仕事です。

しかしもっと細かい部分では、各部位の間の関係が特定のメソッドとパラメータによって定義されているわけではなくて、むしろ各部位はお互いに一般的なメソッドを使って情報をやり取りし、またやり取りするデータとしてもやはり一般的なデータ構造を使っています。それらの一般的なデータ構造の中に、リクエストパラメータ、ヘッダ、属性(アトリビュート)などの情報を入れるのです。

また、従来のGUIアプリケーションと今日の典型的なWebアプリケーションとの、もうひとつの重要な違いは、後者がたったひとつのイベント…HTTPリクエスト…しか認識しないことです。だから、ユーザが今何をしたのか、何を望んでいるのか、といった“実体的なイベント”を知るためにアプリケーションは、リクエストデータをいちいち細かく調べなければなりません。そしてやっと、メニューの××の項目をセレクトしたと分かっても、今度はそのセレクト項目に対する処理をアプリケーション自身が見つけて呼ばなければなりません。言い換えると従来のWebアプリケーションでは、一般のGUIアプリケーションと違って、インタフェイスがそれぞれ独自のイベントを発生することも、また独自のイベント処理をあらかじめ定義しておくことも、できません。これに対して従来のGUIアプリケーションでは、イベントの性質そのものが具体的に細分化されているのです。たとえば「メニュー項目ナニナニがセレクトされた」というイベント、「ＯＫボタンがクリックされた」というイベントなどなど、それぞれのイベントごとに、各イベントを処理するイベントハンドラを書けます。そこで従来からのGUIアプリケーションでは、より細かいレベルでMVCを実装できるのです。

JSFによるMVCの実装は、これまでのWebアプリケーションフレームワークと違って、Swingなどの GUIのフレームワークに似ています。そのことを、図1-2に示します。

図1-2でお分かりのように、JSFではモデルはアプリケーションオブジェクトのプロパティで表さ

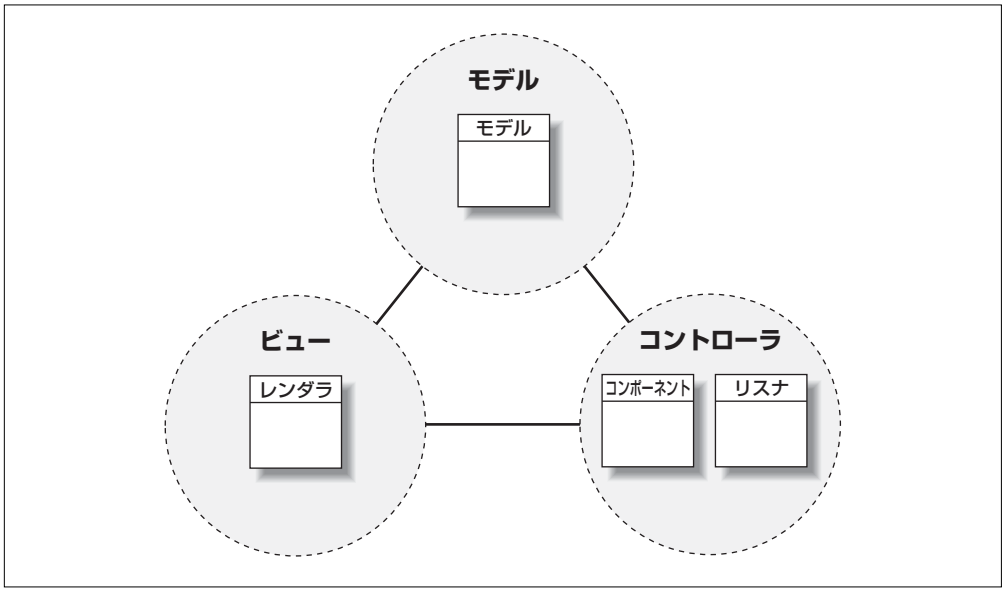

図 1-2 JSF の MVC の設計

れます。たとえば、あるアプリケーションオブジェクトのusernameプロパティは、ユーザの名前というデータを持っています。JSF の UI コンポーネントは、自分が発火するイベント…“値が変わった”イベント、“ボタンがクリックされた”イベントなど…と、それらのイベントをコンポーネントの外部で処理するイベントリスナを宣言します。イベントリスナはイベントを利用する外部からそのコンポーネントにくっつけられ(addXxxListener())、コントローラの一翼を担います。イベントリスナを作ってコンポーネントにくっつける外部とは、コンポーネントの上で起きるイベントと、コンポーネントの制御やビジネスロジックの進行を結び付ける部分です。たとえばあるリスナは、ボタンがクリックされたらファイルをクローズする、といったロジックを実装するでしょう。

イベントリスナが、コンポーネントのプロパティをセットすることもあります。たとえば、あるイベントに対応するリスナは、テーブルの列の表示を調整するでしょう。あるいはまたイベントリスナは、FORMから送信されたデータを処理するアプリケーション側のコードを呼び出すこともあります。そういうアプリ側のコードはたとえば、ユーザが入力したクレジットカードの番号を検査したり、ユーザの入力データに基づいてデータベースを更新したりするでしょう。

ビューを表現するレンダラのクラスは、JSF の UI コンポーネントを実際に画面上に描画します。したがって同じコンポーネントでも、レンダラによって描画のされ方が違います。たとえばあるコンポーネントが、ボタンとして表示されたりリンクとして表示されたりするでしょう。あるいはまた、レンダラを変えて HTML 以外のマークアップ言語を使うこともあるでしょう。

UIコンポーネントのほかにJSFは、ヴァリデータやコンバータといった機能も定義しており、それぞれの機能が明確な役割を担当します。その結果、JSFのユーザインタフェイスは、ひとつひとつがくっきりとしたモジュール性を持ち、したがってメンテナンスしやすく、GUI のアプリケーショ

ンを書き慣れている人は、まるで自分の庭のように感じるでしょう。JSFのこのような性質は、さまざまなサポートツールを作りやすい、ということにも結びつきます。

1.3 JSFの得意技は何？

JSFは、ありとあらゆるWebアプリケーションに適しているわけではありません。動的成分の少ないWebサイト、たとえばデータベースから情報を取り出したり、メニューを動的に生成したり、クッキーの動的な追跡機能をページ上に設けたり、といった程度のアプリケーション、つまり、サイトのコンテンツのメンテナンスとサイトのいろんな部分へのアクセスを円滑にするために動的成分をちょっと使うぐらいのアプリケーションなら、JSFを使うのはおおげさです。JSFに適しているのは、本格的なWebアプリケーションです。本格的というのは、動的成分をちょこっと作って見せるだけでなくて、ユーザとアプリケーションとの対話的なやり取りがつねに大量複雑だ、という意味です。ユーザアクションの少ない単純なWebサイトは、サーブレットとJSPだけで作るほうが楽でしょう。また、JSPとJSTLだけで間に合うものもあるでしょう。

JSFは、今ある技術のどれかを駆逐する性質の技術ではありません。むしろJSFは、アプリケーションのユーザインタフェイスに構造性とメンテナンス性を持たせるために利用する付加的な技術です。それでは、既存のWebアプリケーション技術に対してJSFはどんな機能を提供するのか、その例をいくつか説明しましょう。

1.3.1 JSPとJSFを併用する

さっき見たように、JSFとJSPはよく合います。というか、そもそもJSFの規格は、実装系がJSPをサポートすることと、JSFの標準のUIコンポーネントを表現するカスタムタグの提供を義務づけています。JSPを書き慣れている人は、JSPの中にJSFを容易に混ぜこむことができます。ただし、Javaのコードを1行も書かずにJSPだけでアプリケーションを作れる場合でも、JSFを使うときにはイベントハンドラやアプリケーションのロジックをJavaのクラスとして書かなければなりません。ですから、JSFを使うときにはJavaプログラマが必ず必要です。

1.3.2 Strutsなどのアプリケーションフレームワークと JSFを併用する

Strutsのようないわゆるフレームワークのことを、本書ではアプリケーションフレームワークと呼んでいます。そしてそれに対してJSFを、ユーザインタフェイスのフレームワークと呼びます。あえてそうするのは、両者の目的の違いを強調するためです。

アプリケーションフレームワークの目的は、ひとつのアプリケーション全体の開発をサポートすることです。つまりアプリケーションフレームワークは、アプリケーションの大まかな全体像に関心があります。そこでフレームワークは、交通整理のお巡りさんのように、アッチからコッチへアレを通したり、アソコからドコカへナニを送ったりして、もっぱらアプリケーションの進行役をつとめます。具体的にはたとえば、HTTPのリクエストをリクエストハンドラ(サーブレットの `doGet()` メソッドなど)に送ったり、アプリケーション側からのビューのリクエスト(コンポーネントやレイアウ

トの指示）をリスポンスのレンダラ役（HTML出力）に送ったりします。アプリケーション側のコンポーネントにはそれぞれ名前がついていて、フレームワークはその名前を指定されます。

しかし「アプリケーションフレームワーク」は、細かいことが苦手です。たとえば個々のユーザインタフェイスの実装には無頓着ですし、またユーザのアクションを性格別に仕分けすることもできません。たとえばテーブルの次の行を表示せよというユーザリクエストと商品を注文するリクエストの違いを、フレームワーク自身は認識しません。フレームワークはただリクエストを、リクエストを処理するアプリケーション側のロジックに渡すだけです。そしてその次は、アプリケーション側に指示されたページをリスポンスに収め、JSP、Velocity、XSLTなどのプレゼンテーション層に表示させます。

一方、「ユーザインタフェイスのフレームワーク」のほうは、ユーザインタフェイスの細かい操作が得意で、アプリケーションの実装には無関心です。つまり逆に言うと、アプリケーションフレームワークの場合のように、ユーザインタフェイスの操作にアプリケーションのロジックがいちいち介入しないのです。ユーザインタフェイスのフレームワークは、UIコンポーネントのためのAPI、ユーザのアクションをイベントとして捕捉する仕組み、各イベントに対するイベントハンドラ、そしてデータを視覚的に表現するコンポーネントとビジネスデータとの関係、…これらのものを定義しています。

アプリケーションフレームワークとユーザインタフェイスのフレームワークとの違いが以上のように分かれば、StrutsのようなフレームワークとJSFを十分に併用できることも理解できるでしょう。まず、リクエストはすべてJSFが処理します。そして必要に応じて、アプリケーション側のロジックを呼び出します。ただしJSFはその仕事を、Strutsにやらせます。そしてリスポンスを表示〜描画する段階になると、再びJSFが仕事をします。

ナビゲーションの制御やヴァリデーションなど、一部の機能はJSFとStrutsで重複しています。でも、どの局面でどちらのナビゲーション制御やヴァリデーションを使うかを、簡単に指定できますから、とくに問題はありません。JSFのカスタムタグも、その一部は機能がStrutsと重複していますが、JSFのタグを使ってもStrutsのActionやモデルクラスはほとんどの場合そのまま使えます。

JSFのようなUIコンポーネントをプレゼンテーション層で使用するアプリケーションフレームワークがあります。オープンソースのBarracuda/XMLCやTapestryのほかに、数多くの商用製品もあります。これらのフレームワークをすこし書き換えれば、JSFのコンポーネントをサポートすることも可能ですし、しかもそうしても、アプリケーション側はまったく無関係かつ無変更です。コンポーネントとして何が使われるのか、アプリケーション側はいっさい関知しません。今後多くの商用製品で、そのようなJSF対応の改造が行われるでしょうし、また既存のオープンソース製品も、ユーザのニーズに応じてJSF対応に取り組むでしょう。

1.3.3 エンタプライズ JavaBeans と JSF を併用する

エンタプライズJavaBeans（Enterprise JavaBeans, EJB）などのJ2EE技術は、複数のタイプのクライアント（HTMLブラウザ、WMLブラウザ、通常のGUIアプリケーションなど）を同時にサポートして、高度なセキュリティやトランザクション処理を実現しなければならない、複雑高度なWebア

プリケーションで使われます。しかしこれらの技術は、アプリケーション本体を実装する技術であり、ユーザインタフェイスと直接の関係はありません。したがって、ユーザインタフェイス部分ではJSF をまったく問題なく使えます。

1.4　JSF を勉強していくのに何が必要か？

それでは、本書の例題プログラムや、あなたの自作の JSF アプリケーションを動かすために必要なものを、列挙しましょう：

- インターネットに接続して必要なソフトをダウンロードできるPCまたはワークステーション。
- Java 2 を実装した JDK（Java 2 の SDK）。
- JSP 2.0 対応の Web サーバ†（たとえば Jakarta プロジェクトの Tomcat 5.x）
- JSF 1.0 の実装系（たとえば Sun の参考実装（Reference Implementation））。

本書の例題プログラムはすべて、Tomcat 5 と JSF の参考実装を使ってテストしました。しかしそのほかの JSP 2.0 対応サーバや JSF の実装系でも、問題なく動くはずです。本書の 4 章で、Tomcat のダウンロード、インストール、そしてコンフィギュレーションについて説明します。

JSFをサポートしているツールやサーバは、オープンソース製品と商用製品を含めて、ほかにもいろいろあります。また、IBM、Oracle、Sunなどは、JSFのための開発ツールを提供する、と発表しています。そのほかのベンダも JSF ベースの開発ツールを今後発表するでしょう。ツール関連の最新情報は、Sun の JSF サイトである http://java.sun.com/j2ee/javaserverfaces/ と、James Holmes 氏が個人で作っている JSF 情報サイト http://www.jamesholmes.com/JavaServerFaces/ から得られます。ご自分のアプリケーションを作るときには、それらのツールを試用/評価してもいいですが、しかし本書の例題プログラムを書いて動かすためには、ふつうのテキストエディタとTomcatサーバがあれば十分です。

それでは、JSFをこれから詳しく見ていきましょう。次の章では、実際のアプリケーションの中で JSF をどうやって使うのか、その実例を見ます。

† JSF 1.0 の規格はサーブレット 2.3 と JSP 1.2 を前提しています。しかし本書の例題プログラムでは、JSP 2.0 の新しい機能も使います。