- JSF のユーザインタフェイスでアプリケーションを作る
- アプリケーション本体の開発
- コンポーネントと連結のためのコードを書く
- ユーザインタフェイスのページを作る

2章

JSF アプリケーションの作り方

たとえば航空券を予約するWebアプリケーションを作ることを、想像しましょう。ユーザは、出発地、目的地、日付、等級、航空会社、途中立ち寄り地などを入力します。アプリケーションはこれらの情報を検査して、マッチするフライトの候補一覧を次のページで表示します。ユーザがフライトを指定したら、アプリケーションはそのフライトの予約状況を調べ、空きがあればユーザに席を選ばせ、その席に予約済マークを付け、料金を計算し、クレジットカードの番号を検査し、最後に発券を確定します。

このアプリケーションはコンピュータに詳しくない人も使いますから、分かりやすくて使いやすいユーザインタフェイスが必要です。エラーメッセージが分かりやすいことや、問題が起きたときの対策が簡単にできることも必要です。たとえば目的地を入力するインタフェイスは、最初に国、州、都市名を尋ね、指定された地区にある空港を表示して選ばせるでしょう。また、日付を入力するインタフェイスはカレンダーを表示して日付を選ばせ、フライトを選ばせるインタフェイスはフライト一覧をテーブルで表示するでしょう。これらのインタフェイスは、ユーザからの要望や利用履歴の分析に基づいて改良しやすいように作る必要があります。またアプリケーション本体のコードも、けっこう複雑です。各社のフライトスケジュールや予約状況にリアルタイムでアクセスしますし、クレジットカードの会社にも接続します。それに、一連のトランザクションのセキュリティを完全に確保することも重要です。

このようなアプリケーションは、事前にきめこまかい設計をせずにいきなり作れるものではありません。1章で簡単に説明したMVCの考え方を使って、まずアプリケーションを、ビジネスデータとビジネスロジックを表現する部分(モデルの部分：顧客情報、空港情報、フライト情報、座席情報、などなど)、ユーザインタフェイス(ビューの部分：出発地を入力する入力欄、空港一覧表、などなど)、そして両者を動的につなぐ部分(コントローラの部分)の三部分に分けます。

ビューの部分は、クライアントサイドとサーバサイドのさまざまな方法を使って、いろんな実装が可能です。たとえば入力欄やリストボックスやカレンダー用のテーブルなどのあるHTMLページは、JSP、Velocity、単純なサーブレットなど、Javaの従来のサーバサイド技術を使って表示できます。しかしこれだけ複雑なユーザインタフェイスを従来の方法で作ると、ページのコードの量が多くなり、テーブルのレイアウトのちょっとした変更なども簡単にはできなくなります。

JSFを使ってこのアプリケーションを実装すると、ひとつひとつのユーザインタフェイスをひとつひとつのオブジェクトで表現でき、それらのオブジェクトが、UIのステート(簡易表示か詳細表示か、テーブルに表示する行数、表示の開始位置、など)とアプリケーションのデータ(ユーザが指定したフライトなど)を認識し管理します。ユーザインタフェイスのオブジェクトはまた、プログラマやユーザの指定、あるいはユーザが使っているPCの状態等をもとに、自分の表示を変えるやり方も知っています。簡易表示から詳細表示に変えよ、というボタンをユーザがクリックしたら、そのユーザアクションはイベントとして表現され、ユーザインタフェイスオブジェクト自身またはアプリケーションが提供しているイベントリスナによって処理されます。このように、コンポーネントの上で起きるイベントを軸にアプリケーションのストーリーが進行していく方式は、何年も前からGUIアプリケーションで利用されてきました。この方式を使うと、イベントを列挙して、各イベントのハンドラを書いていくだけですから、どんなに複雑なユーザインタフェイスでも楽に開発できます。そして今、JSF によって、それを Web アプリケーションの開発でも使えるのです。

2.1 JSFのユーザインタフェイスでアプリケーションを作る

JSFを使うとWebの複雑なユーザインタフェイスの開発がどれだけ簡単になるのか、そのことを、ニューズレター[†]の講読申し込みページの例で感じ取っていただきましょう。このアプリケーションには、ユーザがメールアドレスを入力して講読したいニューズレターを選ぶためのFORMと、そのFORM を送信するためのボタンがあります。図 2-1 が、そのユーザインタフェイスです。

ユーザがこのフォームを送信すると、メールアドレスと講読誌名がデータベースに保存されます。そしてアプリケーションの別の部分が、この情報に基づいてニューズレターを送信します。しかしここでは、講読申し込みページだけを細かく見ていきましょう。

JSFを使うアプリケーション開発は、いろんな種類の活動に分かれます。それらの活動の中で、デベロッパであるあなたが担当する役割(role, ロール)は何でしょうか？。開発工程を“役割”という視点で分割してみると、アプリケーション開発の全体の構造がよく見通せるようになります。ただしもちろん、一人の人が複数の役割を担当することもあります。

まず第一に、JSF のフレームワークの実装系が必要です。そこで、第一に挙げるべき役割は “JSF

図 2-1 ニューズレターの講読申し込みフォーム

† 訳者注：日本流に言うと“メルマガ”のようなもの。

のインプリメンタ” です。この役割は主に、Web コンテナ(JSF 対応サーバなど)のベンダが担当するでしょう。次の役割は“ツールのプロバイダ” です。これは、JSFを使ったアプリケーション開発を支援するツールの制作を担当します。この役割を、Webコンテナのベンダが担当することもありますが、開発ツール専門のベンダ、たとえばMacromediaなどがその主な候補です。前にも言ったようにJSF は製品ではなくて規格ですから、いろんな実装が市場で競争し、ユーザはその中から選ぶことになります。

アプリケーションを開発するときにはしかし、JSFの実装系も開発ツールもすでに決まっていることが多いですから、さっきの二つの役割は忘れて、残る役割を検討しましょう。それは、“アプリケーションのデベロッパ”、“コンポーネントの作者”、そして“ページの作者” です。ここでは、ニューズレターの講読申し込みページを実装するときの役割分担を見ていくと、JSFの効能もよく分かると思います。

2.2　アプリケーション本体の開発

アプリケーションのデベロッパという役割は、アプリケーション本体の開発を担当します。言い換えると彼 / 彼女は、ビジネスロジックとビジネスデータを表現するクラスを作ります。

ニューズレターの講読申し込みフォームのためにアプリケーションのデベロッパは、講読者の情報を収める Subscriber というクラスを作ります：

```
package com.mycompany.newsservice.models;

public class Subscriber {
    private String emailAddr;
    private String[] subscriptionIds;

    public String getEmailAddr() {
        return emailAddr;
    }

    public void setEmailAddr(String emailAddr) {
        this.emailAddr = emailAddr;
    }

    public String[] getSubscriptionIds() {
        return subscriptionIds;
    }

    public void setSubscriptionIds(String[] subscriptionIds) {
        this.subscriptionIds = subscriptionIds;
    }
}
```

このSubscriberクラスのフィールドとメソッドには、JavaBeansの規格に沿った名前が付いています。そういうフィールドのことをJavaBeansの用語で“プロパティ”と呼びます。プロパティの値を取得するメソッドの名前は、getに続けてプロパティの名前(先頭が大文字)を書きます。値をセッ

トするメソッドは、setに続けてプロパティの名前(先頭が大文字)を書きます。このような命名規約は、このクラスをJSFのUIコンポーネントのモデルとして使うときにも便利なのです。それについては、すぐあとで説明します。

講読の申し込みや更新が行われると、その情報をデータベースかどこかに保存しなければなりません。アプリケーションのデベロッパは、その仕事を別のクラスにやせるか、それともSubscriberクラスの中でやるか、という判断をするでしょう。ここでは話を簡単にするために、Subscriberクラスの中で情報の保存をすることにしましょう。ただし下のコードでは、情報をデータベースに保存せずに、単純にSystem.outに出力しています：

```
public void save() {
    StringBuffer subscriptions = new StringBuffer();
    if (subscriptionIds != null) {
        for (int i = 0; i < subscriptionIds.length; i++) {
            subscriptions.append(subscriptionIds[i]).append(" ");
        }
    }
    System.out.println("Subscriber Email Address: " + emailAddress +
        "\nSubscriptions: " + subscriptions);
}
```

以上が、この簡単なアプリケーションの本体部分(バックエンドの部分)です。しかし簡単だからといって見過ごさずに、重要なことに気づいてください。アプリケーション本体のクラスはJSFのクラスにかぎらず、ユーザインタフェイスのクラスをひとつも参照していません。ですから逆に言うと、今後どんなタイプのユーザインタフェイスでも使えるのです。

2.3 コンポーネントと連結のためのコードを書く

アプリケーションのデベロッパに続いて、今度はコンポーネントの作者の役割の人が、ユーザインタフェイスのために必要なアプリケーションのコードを書きます。それらは、ユーザインタフェイスとアプリケーション本体部分を結び付けるためのクラス、JSFが最初から提供しているUIコンポーネントを補うための独自のユーザインタフェイス、などです。

図2-2が、このニューズレターアプリケーションが使用するメインのクラスとインタフェイスのクラスです。

この図の中で、Subscriberクラスはアプリケーションのデベロッパがすでに書いたクラスです。コンポーネントの作者は、SubscriberHandlerクラスを書きます。そのほかのクラスはすべて、JSFの実装にあるものです。

UIComponentBaseクラスは、JSFのすべてのUIコンポーネントのベースクラスです。これのサブクラスが、個々のインタフェイス成分、たとえばテキストフィールド、リンク、ボタン、ラベル、メニュー、リストボックスなどなどを表現します。

図2-2の中の、UIInput、UISelectManyなどのクラスは、JSFが最初から提供しているUIコンポーネントです。JSFには、このようなすぐに使えるコンポーネントがいくつも定義されていますが、

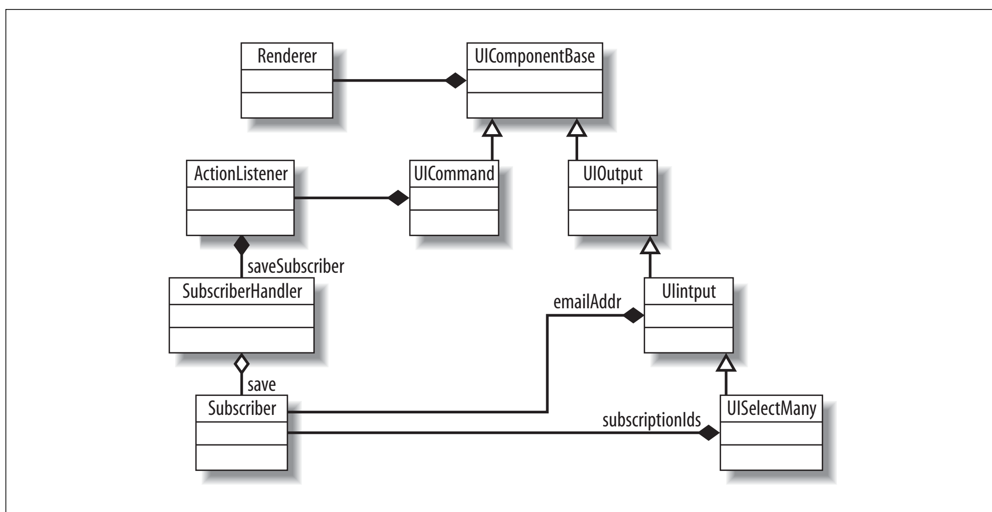

図 2-2　アプリケーションクラス Subscriber と JSF のコンポーネント関連のクラス

もちろんコンポーネントの作者が独自のコンポーネントを書くことも可能です。UIInputクラスは入力欄を表現し、入力された値とアプリケーションのモデル（emailAddrプロパティ）をJSFが結び付けます。同じ仕組みによって、コンポーネントが表示されるときに表示する値をモデルから自動的に取り出すこともできます。

入力用のコンポーネントがリクエストを処理するときには、そのコンポーネントに結び付けられているモデルの値を、リクエストから受け取った値で更新します。図 2-2 の場合には、UIInput（テキストフィールド）とUISelectMany（チェックボックスグループ）の値が、アプリケーションクラスSubscriberのプロパティ（emailAddrとsubscripsionIds）に結び付けられるのです。コンポーネントとアプリケーションのモデルとの間の、このような「値」を軸とする関係のことを、「値結合（value binding, ヴァリューバインディング , 値との結びつき）」と称しています。

コンポーネントは、ユーザのアクション（例：ボタンがクリックされた）に応じてイベントを発火し、そのコンポーネントに登録されているイベントリスナがイベントを処理します（たとえばデータベースを更新します）。しかしJSFのアプリケーションを作るときには、多くの場合、いちいちリスナを書いてコンポーネントに登録する方法ではなく、「メソッド結合（method binding,メソッドバインディング , メソッドとの結びつき）」と呼ばれる簡易な方法†を使うでしょう。メソッド結合は値結合と似ていますが、コンポーネントとアプリケーション本体を、プロパティの値ではなくアプリケーションのメソッドで結び付けます。

この、値結合とメソッド結合は、イベント処理と並んで、JSFの最も重要でしかも便利な仕組みのひとつ、おっと、二つです。

たとえば図 2-2 の中の UICommand というコンポーネントは、そのプロパティのひとつ（20 ページ

† 訳者注：完全な形のイベントリスナではなく、リスナが呼び出すメソッドだけを書けばよい、あとはJSFにおまかせ、という省力化の仕組み。

のリスト2-1に見られるaction属性の値)が、メソッド結合に使われるメソッドです。このコンポーネントがActionEventを発火すると、このコンポーネントのデフォルトのActionListenerが、メソッド結合が指定しているメソッドを呼び出します。そうするとコンポーネントの作者の仕事は、それらのメソッドを書くだけです。デフォルトのActionListenerとは、JSFがこのコンポーネントに自動的に登録している楽屋裏のイベントリスナです。

講読申し込みページのアプリケーションの場合は、コンポーネントの作者が、SubscriberHandlerというクラスの中に、アクションイベントを処理するメソッドを書きました：

```
package com.mycompany.newsservice.handlers;

import com.mycompany.newsservice.models.Subscriber;

public class SubscriberHandler {
    private Subscriber subscriber;

    public void setSubscriber(Subscriber subscriber) {
        this.subscriber = subscriber;
    }

    public String saveSubscriber() {
        subscriber.save();
        return "success";
    }
}
```

このSubscriberHandlerクラスには、二つのメソッドがあります。それは、Subscriberクラスのオブジェクト(アプリケーション本体クラスのオブジェクト)をこのクラスに結び付けるためのset...()メソッドと、SaveボタンのActionEventを処理するメソッドです。そのsaveSubscriber()メソッドは、set...()メソッドで受け取ったアプリケーションオブジェクトのsave()メソッドを呼び出してから、"success"という文字列を返します。本格的なアプリケーションでは、データベースに接続できなかったとか、さまざまなトラブルやエラーに対応して、もっといろいろな文字列を返すでしょう。

アプリケーション本体のクラスのメソッドを呼び出すためにわざわざ新たなクラスを書くなんてくだらない、とも思えます。たしかに、この簡単な例ではそのとおりです。しかし本書の中でこれから見ていくように、もっと複雑なアプリケーションを作るときには、こういうクラスにいろんなことを盛り込めるのです。その主な目的は、アプリケーションオブジェクトにはないものを補って、アプリケーションとJSFのコンポーネントとの間の橋渡しをすることです。

たとえばこの例では、アプリケーションオブジェクトのsave()メソッドはvoidタイプですから値を返しません。それを直接呼んだら、何も返し値は得られません。メソッド結合のためのクラス(SubscriberHandler)のsaveSubscriber()メソッドが、アプリケーションオブジェクトのsave()メソッドを呼び出すだけでなくてStringを返しているのは、アクションイベントを処理するJSF側のメソッドの便宜のためです。JSFはその返し値を見て、次に何をやるかを判断できるのです。ユーザインタフェイスの構造が複雑なアプリケーションでは、こういういろんな返し値を利用できるで

しょう。でもこの話題は後回しにして、コンポーネントの作者の次の仕事に進みましょう。

JSFは、SubscriberやSubscriberHanlderのようなアプリケーションクラスのインスタンスを、faces-config.xmlというコンフィギュレーションファイルの内容に基づいてコンフィギュレーション（構成）します[†]。インスタンスのコンフィギュレーションとはたとえば、ページの作者が値結合やメソッド結合の式の中で使う変数名を、インスタンスの名前として登録することです。その例を、次の節で見ましょう。

コンフィギュレーションファイルを書くことも、コンポーネントの作者の担当です。なぜならそれは、ユーザインタフェイスとアプリケーション本体を結び付ける作業の一環だからです。以下は、JSFのコンフィギュレーションファイルの一部です。この中には、ニューズレター講読申し込みアプリケーションのための宣言がいくつかあります：

```
<faces-config>
  ...
  <managed-bean>
    <managed-bean-name>subscr</managed-bean-name>
    <managed-bean-class>
      com.mycompany.newsservice.models.Subscriber
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
  <managed-bean>
    <managed-bean-name>subscrHandler</managed-bean-name>
    <managed-bean-class>
      com.mycompany.newsservice.handlers.SubscriberHandler
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
      <property-name>subscriber</property-name>
      <value>#{subscr}</value>
    </managed-property>
  </managed-bean>
  ...
</faces-config>
```

このコンフィギュレーションファイルは、今ふうにXMLです。アプリケーションオブジェクトは<managed-bean>成分で宣言します。最初の<managed-bean>成分は、Subscriberクラスのインスタンスをsubscrという変数名で作る、そのスコープはセッションである、と宣言しています。スコープについては4章で説明しますが、この例のようにスコープをセッションにすると、オブジェクトが各ユーザごとに作られ、ユーザがアプリケーションを使っている間（＝講読申し込み手続きをやっている間）ずっと、そのオブジェクトにアクセスできます。

二つめの<managed-bean>成分は、SubscriberHandlerのインスタンスをsubscrHandlerという変数名で宣言しています。この成分には<managed-property>というサブ成分があって、Subscriberという名前のプロパティを宣言しています。このプロパティの初期値が、上で宣言され

† 訳者注：今後この訳書の中では、“コンフィギュレーションする”ではなく、“構成する”と訳す場面が多いでしょう。

ているsubscrなのです。こうして、SubscriberHandlerのインスタンスがSubscriberのインスタンスにリンクされます。

2.4 ユーザインタフェイスのページを作る

Javaのクラスの定義と実装が終わると(たとえそれが初期のプロトタイプのようなものであっても)、ページの作者が仕事にとりかかれます。

ページの作者は、アプリケーションのユーザインタフェイスとして使われるページを書くことが仕事です。そのページはふつう、静的コンテンツ(テキスト、グラフィクス、レイアウトのためのテーブルなど)と動的に生成されるコンテンツの指示が入り混じったテンプレート(ページの基本形を記述している文書)です。ページは一連のUIコンポーネントで表現され、それぞれのコンポーネントがアプリケーションのデータとメソッドに結びついています。静的コンテンツと、コンポーネントから生成される動的コンテンツが、組み合わさってブラウザへ送られます。ユーザがそのページの上のリンクやボタンをクリックすると、そのUIコンポーネントに結びついているメソッドが、クリックによって発生したリクエストを処理します。その結果に基づいて、同じページが再び表示されたり、アプリケーションが別のページを選んでユーザに送ったりします。

前に述べたように、JSFのプレゼンテーション層は差し替え自由ですから、テンプレートの実際の内容はJSFの実装系がサポートしているプレゼンテーション層の種類によって違ってきます。しかしJSF 1.0は、プレゼンテーション層のひとつとしてすべての実装系がJSPをサポートすべし、と定めています。JSPはすでに多くの人が使っていますから、JSFのこの規約はJSFの敷居を低くします。しかしJSPは、それ自身が動的コンテンツを静的テンプレートに加える方式を持っていますから、それがJSFのUIコンポーネントと混在すると混乱が起きるおそれがあります。本書では、このニューズレターのアプリケーションをはじめ、ほとんどのプログラム例でJSPを使いますが、しかし読者はつねに、JSPはJSFにとってひとつのオプションにすぎない、という点を意識してください。また、JSPとJSFの混在がもたらすトラブルを、そんなに心配する必要もありません。本書の中で、それらの問題に出会うたびに対策を説明します。

ともかく、JSPをプレゼンテーション層として使うことに決めたら、ページの作者はJSPのページを書きます。その中に静的コンテンツおよび、JSFのコンポーネントを表現する成分を書いていくのです。リスト2-1は、ニューズレターの講読申し込みフォームのためのJSF成分を加えたJSPページです。

リスト2-1 JSFによる講読申し込みフォームのあるJSPページ(newsservice/subscribe.jsp)

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
  <head>
    <title>Newsletter Subscription</title>
  </head>
  <body>
    <f:view>
```

リスト 2-1　JSF による講読申し込みフォームのある JSP ページ（newsservice/subscribe.jsp）（続き）

```
      <h:form>
        <table>
          <tr>
            <td>Email Address:</td>
            <td>
              <h:inputText value="#{subscr.emailAddr}" />
            </td>
          </tr>
          <td>
            <td>News Letters:</td>
            <td>
              <h:selectManyCheckbox value="#{subscr.subscriptionIds}">
                <f:selectItem itemValue="1" itemLabel="JSF News" />
                <f:selectItem itemValue="2" itemLabel="IT Industry News" />
                <f:selectItem itemValue="3" itemLabel="Company News" />
              </h:selectManyCheckbox>
            </td>
          </tr>
        </table>
        <h:commandButton value="Save"
          action="#{subscrHandler.saveSubscriber}" />
      </h:form>
    </f:view>
  </body>
</html>
```

リスト 2-1 の冒頭には、ご覧のように JSP のカスタムタグライブラリの宣言が二つあります。JSP をあまりよく知らない人も、心配ご無用。本書の 4 章で必要最少限のことを説明します。とにかく JSPでは、このような宣言によって、このページの中で使用する特殊なタグを指定するのです。ここではそれはもちろん、JSFの成分を指定するためのタグです。まず、hというプリフィクス（接頭辞）が付く成分は、HTML で表現される JSF の UI コンポーネントを表現します（h は HTML の略です）。接頭辞 f が付く成分は、ヴァリデータ、イベントリスナなど、UI コンポーネントに付随しているオブジェクトを表します（f は Faces の略です）。

カスタムタグライブラリの宣言に続いて、レイアウトのためのHTML成分と、JSFのUIコンポーネントを表現する JSF 成分があります。最初の <f:view> という成分は、当面忘れてください。その次の<h:form>成分が、JSFのフォームを表すコンポーネントを表現します（HTMLのFORMに相当）。HTML と同様に JSF でも、入力用のコンポーネントはフォームコンポーネントの内側に入れます。

メールアドレスを入力するための入力コンポーネントが、<h:inputText>と書かれている成分です。そのvalue属性は、値結合の式を指定しています。それはこのコンポーネントを、subscrという名前のアプリケーションビーンのemailAddrプロパティに結び付ける式です。次に、ニューズレターの誌名の一覧を <h:selectManyCheckbox> で表します。そのひとつひとつの選択項目は、<f:selectItem>成分としてコンポーネントの中に並べます。<h:selectManyCheckbox>のvalue属性も、値結合の式を指定しています。それはこのコンポーネントを、subscrという名前のオブジェ

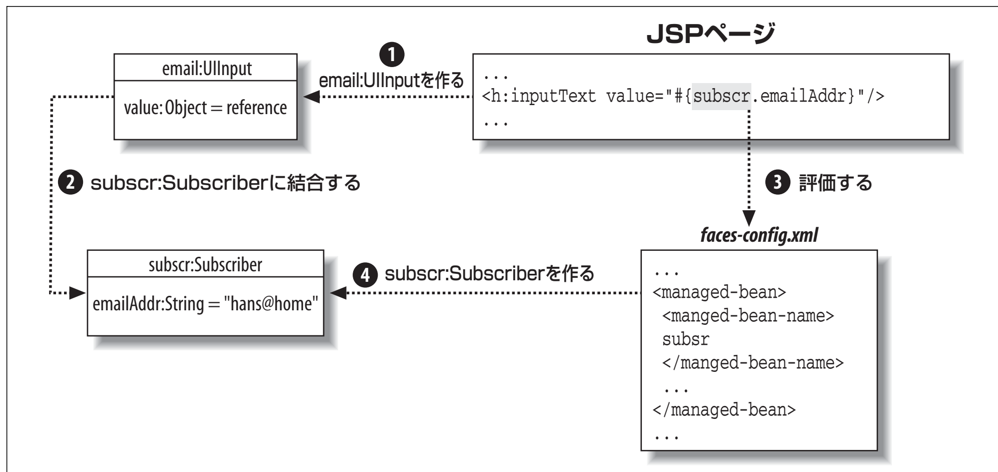

図2-3 リスト2-1のJSPページが最初に処理されるときに作られるオブジェクト

クト(Subscriberビーン)のsubscriptionIdsプロパティに結び付ける式です。

最後の<h:commandButton>成分は、Saveボタンを表現します。そのaction属性は、メソッド結合の式を指定しています。それはこのボタンを、subscrHandlerという名前のSubscriberHandlerビーンの、saveSubscriber()メソッドに結び付ける式です。

値結合とメソッド結合の式はどちらも、#{.....}という形をしています。

このJSPページをユーザがはじめてリクエストとしたときに何が起きるのか、それを図2-3に図解しています。ご覧のように、JSPページの中のJSF成分が指定している値結合やメソッド結合の式と、faces-config.xmlの中のビーンの宣言が共同して、全体の構造を作り上げています。

<h:inputText>成分からはUIInputコンポーネントが作られ、それは値結合の式で指定されているビーンのプロパティに結び付けられます。それから、コンポーネントは画面への表示を指示されます。コンポーネントは値結合の式を評価し、そのビーンがまだ存在しなければfaces-config.xmlの中の情報に基づいてJSFがビーンを作ります。この入力コンポーネントは、値結合の式で指定されているビーンのプロパティから値を取り出し、表示されるHTMLの<input>成分の値として使います。そのほかのJSF成分もこれと同じように処理され、静的コンテンツとJSFのコンポーネントが生成したコンテンツを組み合わせたものが、ブラウザへ送られます。

ユーザが値を入力して送信ボタンをクリックしたら、JSFはそのリクエストを処理するために、まず各コンポーネントに値の取り出しを求めます。それぞれの入力コンポーネントは、その値を使って、自分に結び付けられているビーンのプロパティの値をセットし、またコマンドコンポーネントは、自分に結び付けられているメソッドを呼び出すイベントを発火します。それらのメソッドは通常、データベースの更新などアプリケーション本体の処理を行いますが、この簡単な例題プログラムでは、さきほど見たように値をコンソール(System.out)に書き出すだけです(Subscriberのsave()メソッド, 16ページ)。

次にJSFは、同じJSPページまたは別のページを送り出すためのリスポンスを作ります。イベン

トを処理するメソッドの返し値や、そのほかのコンフィギュレーション項目によって、どのページを送るかが決まります。同じページを送るときは、コンポーネントとアプリケーションオブジェクトがすでに存在していますから、新たなオブジェクトは作られません。その JSP ページは、最初のときとまったく同じように処理されます。

以上は、実行時に起きることのごく簡単な説明です。しかし実際には、これから見ていくように、フォームが送信されるともっといろんなことが起きます。たとえば、入力された値を変換する必要があったり、値の検査（ヴァリデーション）が必要だったり、エラーメッセージをキューに入れたり、ビーンのプロパティを更新せずに同じページを強制的に再表示させたり、などなど。しかしこの章では、できるだけ単純化した形で、JSFのコンポーネントとアプリケーションのコードが連動する様子を見ました。こんな駆け足の見学旅行では、JSFのアーキテクチャもリクエスト処理の詳細もぜんぜん分からない、と途方に暮れているかたもおられるでしょうが、ご心配はいりません。今後の章でその一歩一歩を詳しく見ていけば、すべて明快に分かるようになります。