Oracle® WebCenter Framework

チュートリアル

10*g*（10.1.3.2.0）

部品番号 : E05043-01

2007 年 4 月

ORACLE®

Oracle WebCenter Framework チュートリアル , 10*g*（10.1.3.2.0）

部品番号 : E05043-01

原本名 : Oracle WebCenter Framework Tutorial, 10*g* (10.1.3.2.0)

原本部品番号 : B31072-02

原本著者 : Marcie Caccamo, Rosie Harvey

目次

はじめに

このチュートリアルでは、Oracle WebCenter Suite の主要なコンポーネントの 1 つである Oracle WebCenter Framework について説明します。このチュートリアルで作業を進めるうちに、Oracle JDeveloper と、Oracle WebCenter Framework の新機能をサポートするために追加されたコンポーネントについて理解できます。独自のアプリケーションの構築を始める準備ができたら、『Oracle WebCenter Framework 開発者ガイド』に進むことができます。

注意： このマニュアルの Portable Document Format（PDF）版で URL が 2 行にわかれている場合は、この URL をクリックしてもブラウザに完全な URL データは送信されません。PDF に含まれている URL にアクセスするには、URL をコピーし、ブラウザのアドレス・フィールドに貼り付けます。このマニュアルの HTML 版では、リンクをクリックすればブラウザに直接リンク先が表示されます。

対象読者

このチュートリアルでは、Oracle JDeveloper または Oracle WebCenter Suite に関する予備知識を前提としていません。ただし、次のものに関してある程度理解していることを前提としています。

- Oracle Application Development Framework（Oracle ADF）（目的および基本的なアーキテクチャ）
- Oracle ADF Faces
- Java

このチュートリアルは、WebCenter アプリケーションを構築しようとする開発者、または Oracle ADF を使用してアプリケーション内にカスタマイズ機能を構築しようとするアプリケーション開発者を対象としています。

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様にもオラクル社の製品、サービスおよびサポート・ドキュメントを簡単にご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML 形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社のアクセシビリティについての詳細情報は、Oracle Accessibility Program の Web サイト `http://www.oracle.com/accessibility/` を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

スクリーン・リーダーは、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧だけを行に記述する必要があります。しかし JAWS は括弧だけの行を読まない場合があります。

外部 Web サイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しない Web サイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらの Web サイトのアクセシビリティに関しての評価や言及は行っておりません。

Oracle サポート・サービスへの TTY アクセス

アメリカ国内では、Oracle サポート・サービスへ 24 時間年中無休でテキスト電話（TTY）アクセスが提供されています。TTY サポートについては、(800)446-2398 にお電話ください。

関連ドキュメント

Oracle WebCenter Framework の詳細は、次のドキュメントを参照してください。

- 『Oracle WebCenter Framework 開発者ガイド』
- 『Oracle WebCenter Framework エラー・メッセージ・ガイド』

Application Development Framework の詳細は、次のドキュメントを参照してください。どちらのドキュメントも、Oracle Technology Network（OTN）（`http://www.oracle.com/technology/index.html`）で入手できます。

- 『Oracle Application Development Framework チュートリアル』
- 『Oracle Application Development Framework 開発者ガイド』

表記規則

このマニュアルでは次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連する Graphical User Interface 要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック	イタリックは、ユーザーが特定の値を指定するプレースホルダ変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、または入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

オラクル製品サポートの購入方法、および Oracle サポート・サービスへの連絡方法の詳細は、次の URL を参照してください。

`http://www.oracle.co.jp/support/`

製品マニュアル

製品のマニュアルは、次の URL にあります。

`http://otn.oracle.co.jp/document/`

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

`http://www.oracle.co.jp/education/`

その他の情報

オラクル製品やサービスに関するその他の情報については、次の URL から参照してください。

`http://www.oracle.co.jp`
`http://otn.oracle.co.jp`

注意： ドキュメント内に記載されている URL や参照ドキュメントには、Oracle Corporation が提供する英語の情報も含まれています。日本語版の情報については、前述の URL を参照してください。

第 I 部

WebCenter Framework チュートリアルの概要

第 I 部の内容は次のとおりです。

1

Oracle WebCenter Suite の概要

この章では、Oracle WebCenter Suite の概要を示し、Oracle WebCenter Suite を使用してサービス指向アプリケーションの機能を拡張する方法について説明します。Oracle WebCenter Suite では、より高度な通信、コンテンツ管理機能、カスタマイズおよび拡張検索サポートをユーザーに提供する、アプリケーションと統合可能な各サービスが用意されています。さらに重要なことに、Java Server Faces アプリケーション内でポートレットやコンテンツを消費する機能、宣言型セキュリティ、ライフサイクル管理ツールなどの必要不可欠な機能を提供する開発フレームワークが用意されています。

この章では、次の重要なトピックについて説明します。

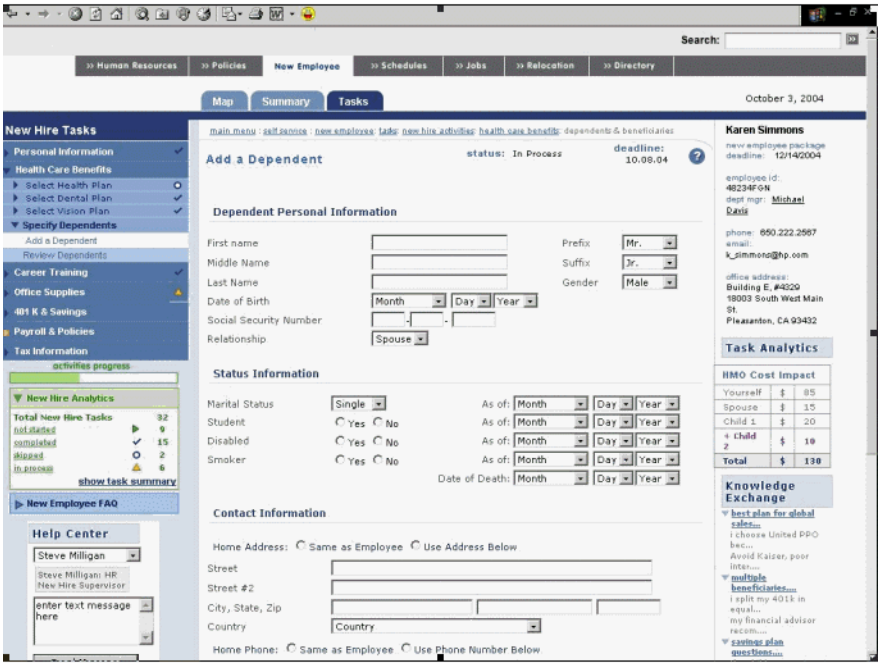
- Oracle WebCenter Suite の概要
- このチュートリアルで学習する内容

この章を読み終えると、独自の Java EE 5 アプリケーションの構築を開始できます。

Oracle WebCenter Suite の概要

Wiki、RSS、ブログなどの主要なテクノロジによって世界中の個々人に能力が与えられ、インターネットの展望が変化する中、ユーザーは、取引を簡素化するアプリケーションをますます求めるようになりました。取引を簡素化するための 1 つの方法は、特定の作業に対応するためにユーザーが必要とするすべての要素をアプリケーションそのものに組み込むことです。たとえば、図 1-1 に示した例で考えてみます。

図 1-1 サンプル・アプリケーション



この例では、会社に入ったばかりのユーザーが、会社の保険契約に扶養家族を追加できるアプリケーションを使用しています。取引そのものの周りに、次のように、ユーザーの理解を助ける追加のコンテキストがあることに注意してください。

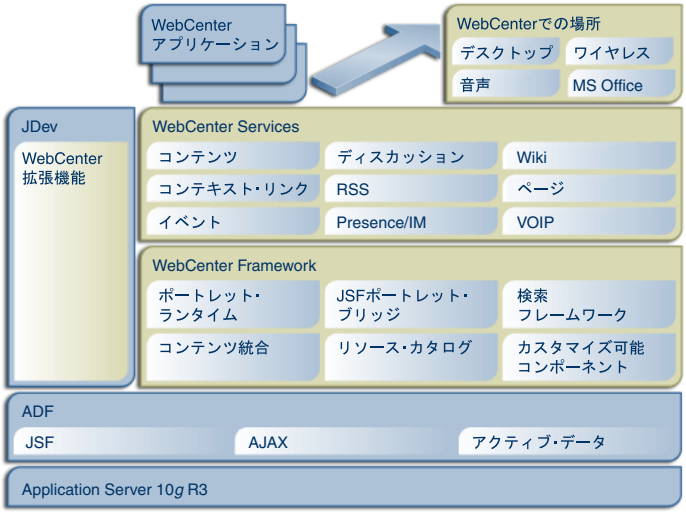
- 左上隅の「New Hires Tasks」には、新しい会社に慣れるという大きなプロセスにおけるユーザーの現在の位置を示すアクティビティ・ガイドがあります。ユーザーの次の作業も示されています。このようなタイプのプロセス編成によって、ユーザーは複数の手順からなるフロー全体を迅速かつ簡単に進むことができます。
- 作業およびプロセスの分析手法によって、ユーザーは、プロセスにおける自分の位置と、選択した内容が自己に及ぼす影響を知ることができます。この例では、右側の「Task Analytics」に、これまでに選択した給付の内容の全体的なコストの影響が示されます。
- 左下の「Help Center」には、一般的な質問をすぐに参照できる最新の FAQ と、FAQ に記載されていない質問をユーザーが質問できるヘルプ・センターへのダイレクト・チャット・リンクがあります。ここでも、ユーザーは取引のコンテキストから離れずにヘルプを受けることができます。
- 右下の「Knowledge Exchange」には、現在の作業に関連するドキュメントが示されます。これらのドキュメントは企業リポジトリ内に格納されており、様々な受給者および扶養家族のシナリオに対する詳細なアドバイスを提供します。

Oracle WebCenter Suite までは、このような種類のアプリケーションを構築することは非常に冗漫なプロセスでした。たとえば、（API での必要に応じて）Java Content Repository（JSR 170）内を参照するためのポートレットの作成に使用される受給者シナリオを利用することが可能でした。Oracle WebCenter Suite では、サービス指向アーキテクチャ（SOA）の概念を採用することで、これまでユーザーに必要なビジネス・コンポーネントを提供するために必要であったフロントエンドの労力を軽減しています。Oracle WebCenter Suite は SOA だけでなく

Java Content Repository（JSR 170）やその他の業界標準に準拠しているため、広範な種類のプラグ・アンド・プレイ製品、ツールおよびサービスを使用でき、ユーザーが求めるアプリケーションを構築することが容易になります。

図 1-2 に、Oracle WebCenter Suite の機能を示します。[1]

図 1-2 Oracle WebCenter Suite



これから、これらの構築ブロックについて詳しく説明します。

WebCenter Framework

WebCenter Framework は、追加の統合オプションとランタイム・カスタマイズ・オプションを提供することによって Java Server Faces（JSF）環境を補強します。つまり、これまでポータル製品に組み込まれていた機能を JSF 環境のファブリックに直接統合します。これによって、ユーザーの人為的な障壁が取り除かれ、図 1-1 に示したようなコンテキスト・リッチなアプリケーションを開発するための基礎が提供されます。

ポートレットの構築および消費

ポートレットを使用すると、Web やデータベースなどからのデータをアプリケーションに取り込むことができます。Oracle JDeveloper を使用すると、JSR 168 または WSRP 互換のポータルによって消費される標準ベースのポートレットを独自に作成できます。Oracle Application Server Portal Developer Kit（PDK）は機能拡張され、WSRP 2.0 で定義されている拡張的なポートレット機能を Java Portlet Standards API の構造内でサポートできるようになりました。WebCenter アプリケーションから、JSR 168、WSRP 1.0、WSRP 2.0 または Oracle PDK-Java ポートレットをすべて同じアプリケーション内あるいは同じページ内で消費できます。

Preconfigured OC4J を介して使用できる、事前構築済のポートレットがいくつかあります（Preconfigured OC4J は JDeveloper を介して自動的に使用可能になります）。このような 2 つのポートレット（OmniPortlet および Web クリッピング）を使用すると、ユーザーは独自のデータを収集できます。これに対し、リッチ・テキスト・ポートレットを使用すると、ユーザーは独自の告知や掲示を公開できます。これらのポートレットは、ページ上にドロップすることにより、ユーザーに対して使用可能にできます。あるいは、これらのポートレットを自分で使用して、ユーザーのニーズに応じた特定のポートレットを作成することもできます。

- OmniPortlet: ユーザーが様々なレイアウトを使用して様々なソースからデータを簡単に公開できるようにするポートレットです。OmniPortlet では、ユーザーがスプレッドシート

[1] 示されているコンポーネントの中には、Oracle WebCenter Suite の初期のリリース（Presence/IM、Discussions および Wiki）では使用できないものもあります。この章では、Oracle WebCenter Framework のこのリリースに関連するコンポーネントについて説明します。

（値間を特定文字で区切ったもの）、XML、Web サービス、既存の Web ページ上のアプリケーション・データなど、ほぼあらゆる種類のデータソースを使用できます。データを取得した後、ユーザーは箇条書きリスト、グラフ、HTML などのレイアウトを使用して、OmniPortlet を書式設定できます。

開発者は、このツールを使用して、ユーザーのデータを収集および書式設定できます。たとえば、従業員ディレクトリを作成する場合は、ユーザー消費用に OmniPortlet をページ上に配置します。配置されたポートレットは、他の人がそれぞれのアプリケーションで使用できるように、JDeveloper のコンポーネント・パレットを介して使用可能になります。

- Web クリッピング : 技術的な専門知識をまったく必要としない、非常に使いやすいウィザードです。ユーザーはクリップする Web コンテンツを見つけ、ウィザードを使用して、そのコンテンツをグラブしてアプリケーション内に表示するだけです。元のサイトの Web コンテンツが更新された場合は、ユーザーの関連付けられた Web クリッピングも更新されます。
- リッチ・テキスト・ポートレット : ユーザーが独自の告知や放送を公開できるようにするツールです。リッチ・テキスト・ポートレットをページ上に配置すると、実行時に、権限のあるユーザーは、表示テキストの挿入、更新および書式設定に必要なすべてのリッチテキスト編集ツールにアクセスできます。

カスタマイズ可能なコンポーネント

WebCenter Framework には新しい JSF コンポーネントが備わっており、これを使用すると、開発者はアプリケーションをカスタマイズ可能にすることができます。これらの新しいコンポーネントはコンテナとして機能します。開発者は、このコンテナに別の Faces ビュー・コンポーネントやポートレットをドロップできます。これらの機能が設定されていれば、開発者は、ページ上でコンポーネントの最小化 / 最大化、非表示 / 表示、または移動を行うことにより、ほぼあらゆる JSF ページをカスタマイズできます。

コンテンツ統合

たとえば、Oracle Content DB や OracleAS Portal などのコンテンツ管理システム（またはファイル・システム上）に存在するデータをアプリケーションで使用可能にするとします。WebCenter Framework には、そのコンテンツにアクセスするために必要な JCR アダプタが備わっています。JDeveloper を使用すると、JCR データ・コントロールを構築してコンテンツをグラブし、様々な表示モードでのページ上にドロップできます。また、WebCenter Framework には Oracle Drive が付属しており、これを使用すると、OracleAS Portal リポジトリをデスクトップ上にツリー構造として表すことができます。

アプリケーションの保護

WebCenter Framework に備わっている ADF 拡張要素を使用すると、アプリケーション全体、アプリケーション内のページ、またはカスタマイズ可能コンポーネントにより提供された個々のアクションに対して、セキュリティを定義できます。このチュートリアルでは、簡単なログイン・ページを作成して基本的な権限を複数のユーザーに割り当てる方法を学びます。

多くの場合、電子メールなどの独自の認証メカニズムを備えた既存のアプリケーションを活用すると効果的です。WebCenter Framework では、外部アプリケーション・ウィザードを使用してこれらのアプリケーションを埋め込むことができます。詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

ライフサイクル全体にわたるアプリケーションの管理

WebCenter Framework は、複数のツールを使用することにより、アプリケーションの構築、デプロイおよび移行に必要な時間を短縮します。

- 開発フレームワーク : Oracle JDeveloper および ADF には、アプリケーションの構築と更新に必要なツールおよびフレームワークが備わっています。ポートレット、コンテンツおよびカスタマイズの機能を WebCenter アプリケーションに追加するには、ソースまたは WYSIWYG 環境のいずれかに適切なオブジェクトをドラッグ・アンド・ドロップするだけです。WebCenter Framework には、テストおよびデバッグのフェーズを簡素化するために、ポートレット・カスタマイズ、コンテンツおよびページ・カスタマイズをパッケージ化して（付属のスタンドアロン OC4J などの）J2EE コンテナに移行するデプロイメント・プロファイル（WebCenter アプリケーション WAR）が組み込まれているため、本番サーバーにデプロイする前にアプリケーションをテストしてデバッグできます。
- エンタープライズ・デプロイメント : アプリケーションを本番環境にデプロイする準備ができると、Oracle WebCenter Framework のデプロイ前ツール・パッケージによって、ポートレット・カスタマイズがパッケージ化されて本番の場所に移行され、コンテンツ・リポジトリへのポインタが変更され、アプリケーションがメタデータ・サービスの本番の場所を指しているか確認されます。デプロイ前ツールの処理が完了すると、ターゲットの EAR ファイルが生成されます。この EAR ファイルは、Enterprise Manager を使用して最終的な場所にデプロイできます。
- 標準ベースの管理 : ブラウザベースのツールによって、管理者は、WebCenter アプリケーションをデプロイ、構成および管理できます。また、業界標準ベースの JMX 方式に基づいて構築されたツールによって、管理者はきめ細かな制御を行い、状態ステータス、パフォーマンスおよび評判に関する監視メカニズムを利用できます。また、（単一の Oracle Application Server コンテキスト内で）経時的な履歴パフォーマンスおよびステータス・レポートを取得するためのツールも提供されています。WebCenter アプリケーションのメトリックは、使い慣れた Application Server Control の監視および管理インタフェースを使用して配信されます。

WebCenter Services

WebCenter Services には、次のような様々なコンテンツ管理、検索および通信のサービスがあります。

- Oracle WebCenter Services のデフォルト・コンテンツ・リポジトリである Oracle Content Database（Oracle Content DB）。Oracle Content DB は、ユーザーが Web 経由で、またはデスクトップ・アプリケーションからコンテンツを管理できる、本格的なコンテンツ管理システムです。サービス指向環境でエンタープライズをコンテンツ対応にするための、すぐに使用できる豊富な Web サービス・ライブラリが提供されています。Oracle Content DB を使用すると、次のことが可能です。
  - ビジネス・プロセスのコンテキストで適切なコンテンツにセキュア・アクセスすることにより、個人およびチームの生産性を向上させます。
  - 情報の損失やリーガル・ディスカバリなど、情報コンテンツに伴うリスクを軽減します。
  - ビジネス・プロセスの適応性を促進します。
  - コンテンツの強化によって IT コストおよび管理コストを削減します。

  Oracle Content DB は、機能が限定されたファイル・サーバーと、幅広く使用されている専門的で高価かつ複雑なコンテンツ管理アプリケーションとの間のギャップを埋める製品です。
- Oracle Secure Enterprise Search はクローラベースのサービスであり、幅広いソース（様々なファイル書式の索引付きデータ、リアルタイム・データ、構造化データおよび非構造化データ）の検索を可能にします。Oracle Secure Enterprise Search を使用すると、会社の情報リポジトリで関連ドキュメントを短時間で検索できます。

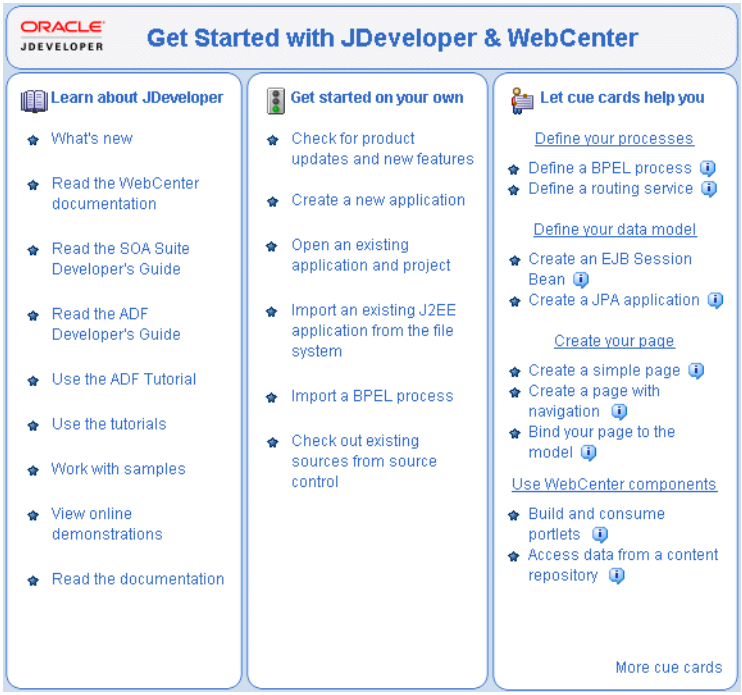
- 人と人とを接続し、通信を促進するための通信サービス。次のようなサービスがあります。
  - インスタント・メッセージング : オーディオおよびビデオ・フィード、ファイル交換、その他様々な機能を介して、ユーザーがアイデアを自由に交換することを可能にします。
  - プレゼンス・サーバー : プレゼンスは、ある人が個々人または自分のステータスにサブスクライブしているアプリケーションに対して使用可能であるかどうかの情報を提供します。チャットやその他のリアルタイム・サービスは、関連付けられたユーザー・インタフェースから開始できます。
  - ディスカッション・フォーラム : 情報、質問およびコメントを共有するためのインタラクティブなメッセージ・ボード。
- Wiki は、ユーザーが Web ブラウザを使用して Web ページ・コンテンツを自由に編集および作成できるようにするサーバー・ソフトウェアです。このように簡単に対話および操作できることから、Wiki はコラボレーティブな通信を行うための効果的なツールとなっています。

Oracle JDeveloper

Oracle JDeveloper は、Java、XML、Web サービスおよび SQL の最新の業界標準を使用してサービス指向アプリケーションを構築するための統合開発環境（IDE）です。Oracle JDeveloper では、アプリケーションのモデリング、コーディング、デバッグ、テスト、プロファイリング、チューニングおよびデプロイを行うための統合的な機能を備え、完全なソフトウェア開発ライフサイクルをサポートしています。Oracle JDeveloper の視覚的で宣言的なアプローチと Oracle Application Development Framework（ADF）がともに機能することにより、アプリケーション開発は簡素化され、日常的なコーディング作業が減ります。たとえば、ボタン、値リスト、ナビゲーション・バーなどの多数の標準的なユーザー・インタフェース・ウィジェットのコードが事前にパッケージ化されています。コンポーネント・パレットから適切なウィジェットを選択して、アプリケーション内にドロップするだけです。

このチュートリアルを読み進むうちに、Oracle JDeveloper とそのメリットを理解できます。Oracle JDeveloper の詳細は、Oracle JDeveloper の「ヘルプ」メニューからアクセス可能な Oracle JDeveloper の開始ページ（図 1-3）より、多数ある参考資料の 1 つにアクセスしてください。

図 1-3 Oracle JDeveloper の開始ページ



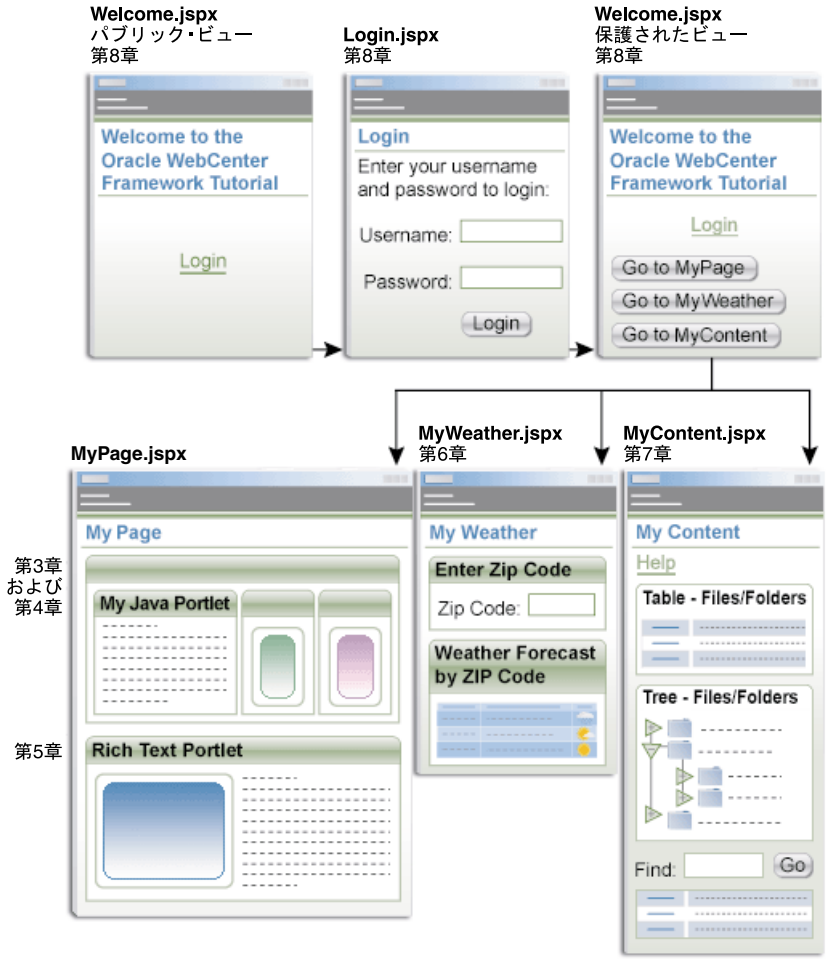
このチュートリアルで学習する内容

このチュートリアルでは、JDeveloper を使用して、次の 5 つの簡単なページを含むアプリケーションを構築します。

- ようこそページ : パブリック・ユーザーにはパブリック・コンテンツを表示し、認証済ユーザーにはセキュア・コンテンツを表示します。
- ログイン・ページ : このページを使用して、ユーザーの自己認証を可能にする方法の基礎を学びます。
- MyPage: このページ上に、ファイル・システムからの Java ポートレット、リッチ・テキスト・ポートレットおよびイメージを配置します。
- MyContent: このページ上に、ファイル・システムからのコンテンツを公開します。
- MyWeather: このページを使用して、OmniPortlet および Parameter Form ポートレットの使用方法と、この 2 つのポートレット間の通信を有効にする方法を学びます。

次の図（図 1-4）に、ページ間の論理的なフローを示します。

図 1-4 ページ間のフロー



これらのページは、図に示された順序で開発するわけではないので注意してください。最初に、最も基本的なページである MyPage を開発してから、徐々に複雑なトピックへと進みます。

このチュートリアルは、各章を記載された順序どおりに完了することを想定しています。各章に依存関係があるため、異なる順序で完了すると、リソースが存在しなかったり、エラーが発生することがあります。このチュートリアルは、次の順序で進めます。

- 第 2 章「チュートリアルを始める前に」で、このチュートリアルの手順を完了するために、あらかじめ実行する必要のある作業を示します。必ず、この章に示されている手順をすべて完了してください。
- 第 3 章「最初のポートレットの構築およびテスト」で、基本的なページである MyPage を構築し、簡単なポートレットを追加してから、より高度なロジックを含むようにポートレットを拡張する方法を示します。
- 第 4 章「ページのカスタマイズ」で、アプリケーションのカスタマイズを可能にする方法について概説します。このレッスンでも、引き続き MyPage を使用します。
- 第 5 章「リッチ・テキスト・ポートレットの追加」で、リッチ・テキスト・ポートレットを MyPage に配置する方法を示します。
- 第 6 章「ポートレット間の通信」で、新しいページ MyWeather を作成し、Parameter Form ポートレットおよび OmniPortlet をこのページに追加します。また、2 つのポートレット間にパラメータを設定して簡単な通信を有効にする方法についても学びます。
- 第 7 章「ページへのコンテンツの追加」で、ファイル・システムからのコンテンツを MyContent というページに追加する方法を示します。また、ページに検索フォームを追加する方法も学びます。
- 第 8 章「セキュリティの設定」で、ようこそページおよびログイン・ページを作成することによって、セキュリティを実装する方法を学びます。MyPage、MyWeather、MyContent、およびようこそページのセキュア・コンテンツにアクセスできるのは、権限のあるユーザーのみです。
- 第 9 章「WebCenter アプリケーションのデプロイ」で、サンプル・アプリケーションをデプロイする手順を示します。

まだ JDeveloper を使用したことがない場合も、このチュートリアルを終えれば、このツールの基本的な用途および機能を把握できるはずです。もちろん、JDeveloper は、その基礎となるフレームワークである ADF と同様に、このチュートリアルでは簡単にしか説明していない大きなメリットを提供します。本格的に開発を開始する前に、この両方の製品についてさらに学習する意欲を持つはずです。非常に有用な 2 つのリソースを次に示します。どちらも、Oracle Technology Network（`http://www.oracle.com/technology/`）で入手可能です。

- 『Oracle Application Development Framework チュートリアル』
- 『Oracle Application Development Framework 開発者ガイド』

それでは、始めましょう。

2

チュートリアルを始める前に

この章では、Oracle JDeveloper の適切なバージョンと、このチュートリアルのレッスンを完了するために必要ないくつかのファイルをダウンロードする方法を示します。

WebCenter 拡張機能が含まれた Oracle JDeveloper のダウンロード

このチュートリアルを完了するには、Oracle JDeveloper Studio Edition（10.1.3.2.0 以上）にアクセスできるか、この製品がインストールされている必要があります。このリリースには、WebCenter アプリケーションの構築に必要なすべての機能が含まれています。

Oracle JDeveloper Studio Edition（10.1.3.2.0）は、次の Oracle Technology Network からダウンロードできます。

`http://www.oracle.com/technology/products/jdev/index.html`

任意のディレクトリ（このチュートリアルでは *JDEVHOME* と表記）に解凍します。

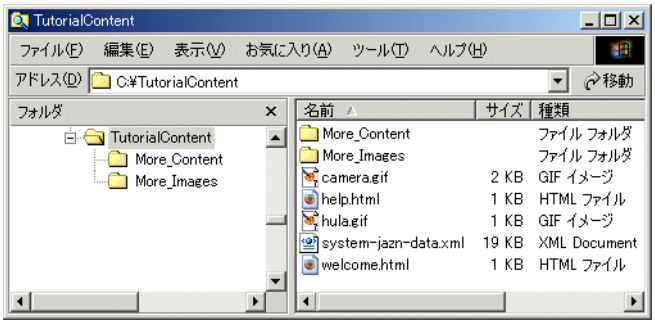
サンプル・チュートリアル・ファイルのダウンロード

このチュートリアルでは、様々な時点で、アプリケーションに特定のコンテンツおよびイメージを含めるように要求されます。この素材は `Zip` ファイルに含まれています。このファイルをダウンロードするには、次の手順を実行します。

1. ブラウザを開き、「アドレス」フィールドに `http://www.oracle.com/technology/products/webcenter/files/webcentertutorialcontent.zip` を入力します。
2. zip ファイル（`webcentertutorialcontent.zip`）を開くには、**「開く」**をクリックします。
3. ファイルをローカル・ドライブ（C など）に解凍します。

   図 2-1 に、`C:¥TutorialContent` に解凍されたファイルを示します。

図 2-1 解凍されたサンプル・コンテンツ Zip ファイル



4. サンプル・コンテンツを格納した場所を書き留めます。チュートリアルでの作業中、何度かこの場所にアクセスする必要があります。

次に、第 8 章「セキュリティの設定」を完了するために必要なユーザー・データを設定します。

サンプルの system-jazn-data.xml ファイルのコピー

ダウンロードしたチュートリアル（webcentertutorialcontent.zip）には、サンプルの system-jazn-data.xml ファイルが含まれています。このファイルには、第 8 章「セキュリティの設定」を完了するために必要なユーザー・データが含まれています。サンプル・データを使用するには、サンプルの system-jazn-data.xml ファイルを複数の Oracle JDeveloper の場所にコピーする必要があります。これらの手順でファイルのコピー先を示します。また、ファイルをコピーする前に、元のファイルをバックアップしておくことをお薦めします。

このファイルは、チュートリアルを簡単にするために便宜上提供されているものです。通常、アプリケーションを構築するときは、JDeveloper を介して追加したユーザー、ロールおよびポリシーが失われる可能性があるため、既存の system-jazn-data.xml を上書きしません。

注意： JDeveloper でセキュアなアプリケーションをすでに構築中で、独自のユーザー・データを system-jazn-data.xml に移入した場合は、チュートリアルのユーザーおよびロールを、現在のユーザー・データと共存できるように最初から定義することもできます。その場合は、ここに示されたサンプルの system-jazn-data.xml ファイルをコピーしないでください。
付録 A「チュートリアルの ID ストアの設定方法」のすべての手順を完了してから、第 8 章「セキュリティの設定」を開始してください。

サンプルの system-jazn-data.xml ファイルをコピーするには、次のようにします。

1. サンプル・チュートリアル・コンテンツを解凍したディレクトリで、サンプルの system-jazn-data.xml ファイルを見つけます。たとえば、C:¥TutorialContent などです。
2. このチュートリアルの目的で変更を行う前に、次の場所にある system-jazn-data.xml ファイルをバックアップします。
   - **Oracle JDeveloper の埋込み OC4J:**
     JDEVHOME¥jdev¥system¥oracle.j2ee.10.1.3.xx.xx¥embedded-oc4j¥config¥system-jazn-data.xml

     JDeveloper を最初に開いたときに、システム・ディレクトリが作成されます。JDEVHOME¥jdev ディレクトリ内にシステム・フォルダが見つからない場合は、JDeveloper を起動して再び停止した後、この手順を実行してください。
   - **Oracle JDeveloper:**
     JDEVHOME¥j2ee¥home¥config¥system-jazn-data.xml
3. サンプルの system-jazn-data.xml ファイルを次のディレクトリにコピーします。
   - JDEVHOME¥jdev¥system¥oracle.j2ee.10.1.3.xx.xx¥embedded-oc4j¥config

     このファイルをコピーすると、ユーザー・データが JDeveloper の埋込み OC4J に対して使用可能になります。
   - JDEVHOME¥j2ee¥home¥config

     このファイルをコピーすると、ユーザー・データが Oracle JDeveloper 認可エディタに対して使用可能になります。詳細は、第 8 章を参照してください。

これで、最初のレッスンである第 3 章「最初のポートレットの構築およびテスト」の「手順 1: JSR 168 Java ポートレット・ウィザードの使用」を開始できます。

第 II 部

実践的な例

第 II 部の内容は次のとおりです。

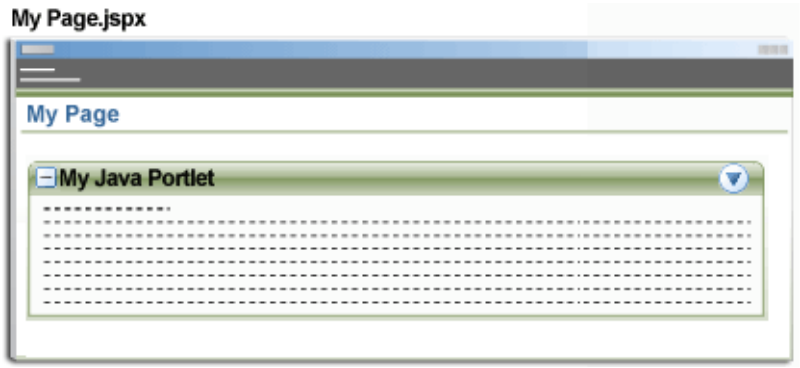
3

最初のポートレットの構築およびテスト

このレッスンでは、最初のポートレットを構築する方法を学びます。ポートレットを作成した後は、ポートレットをテストし、簡単なページを作成してから、ポートレットをページ上にドロップします。次に再びポートレットをテストしてから、ポートレットにいくつかのロジックを追加します。このレッスンを終えると、簡単なポートレットを構築およびテストするための手順を理解できます。

図 3-1 に、このレッスンを終えた時点での完成物の概念的な図を示します。MySample という名前の WebCenter アプリケーションに、MyPage という名前のページ（MyPage.jspx）が含まれています。また、MyJavaPortlet という名前のポートレットを作成します。このポートレットは MyPage に配置してからカスタマイズします。

図 3-1 レッスン 3 を終えた時点での MyPage.jspx



概要

次の手順で、ポートレットを構築してテストします。

- 手順 1: JSR 168 Java ポートレット・ウィザードの使用
- 手順 2: 接続の設定
- 手順 3: ポートレットのデプロイ
- 手順 4: JSF ページの作成
- 手順 5: Oracle WebCenter Framework へのポートレットの登録
- 手順 6: ポートレットのテスト
- 手順 7: ポートレットへの簡単なロジックの追加

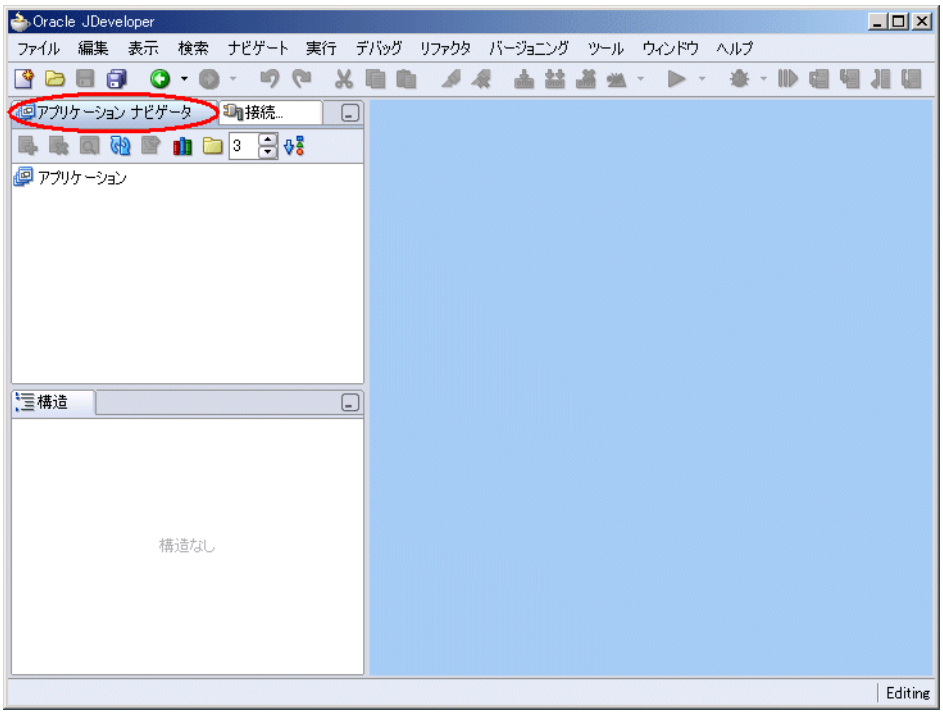
このプロセスは、JDeveloper を使用してどのようなタイプのポートレットを構築するかにかかわらず、だいたい同じです。

手順 1: JSR 168 Java ポートレット・ウィザードの使用

ポートレットを作成する前に、ポートレットのコンテナとなるアプリケーションを作成する必要があります。アプリケーションを作成するには、次のようにします。

1. JDeveloper がインストールされている場所から **jdev.exe** をダブルクリックすることにより、JDeveloper を起動します。
2. 「今日のヒント」が表示されたら、**「閉じる」**をクリックして閉じます。
3. 「アプリケーション・ナビゲータ」タブが選択されていることを確認します（図 3-2）。

図 3-2 JDeveloper でのアプリケーション・ナビゲータ



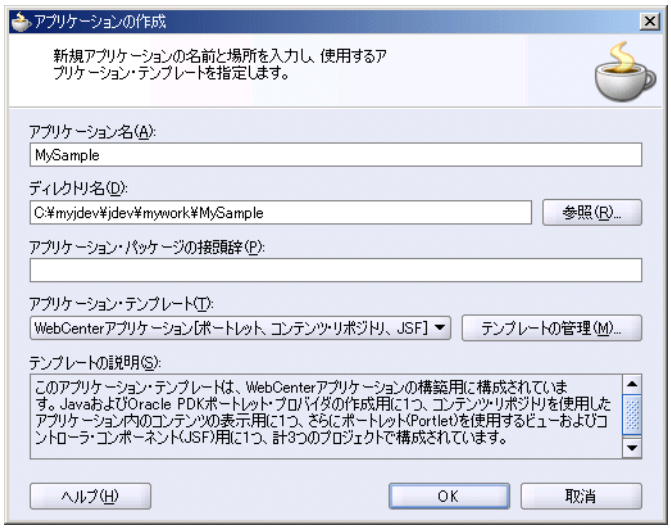
4. アプリケーション・ナビゲータで、**「アプリケーション」**を選択して右クリックします。
5. **「新規アプリケーション」**を選択します。
6. 「アプリケーション名」フィールドに `MySample` を入力します。

7. MySample のデフォルト場所を受け入れ、書き留めておきます。「手順 3: ポートレットのデプロイ」で、この場所が必要になります。

8. 図 3-3 に示す「アプリケーション・テンプレート」プルダウンから、**「WebCenter アプリケーション [ポートレット、コンテンツ・リポジトリ、JSF]」**を選択します。

   このテンプレートによって、必要なプロジェクトが作成され、WebCenter アプリケーションに適用可能なオプションのみが表示されるようにターゲットの JDeveloper が設定されます。

図 3-3 新しいアプリケーションの作成



9. **「OK」**をクリックします。

   アプリケーション・ナビゲータ（図 3-4）で、WebCenter アプリケーションが次の 3 つのプロジェクトで構成されていることがわかります。

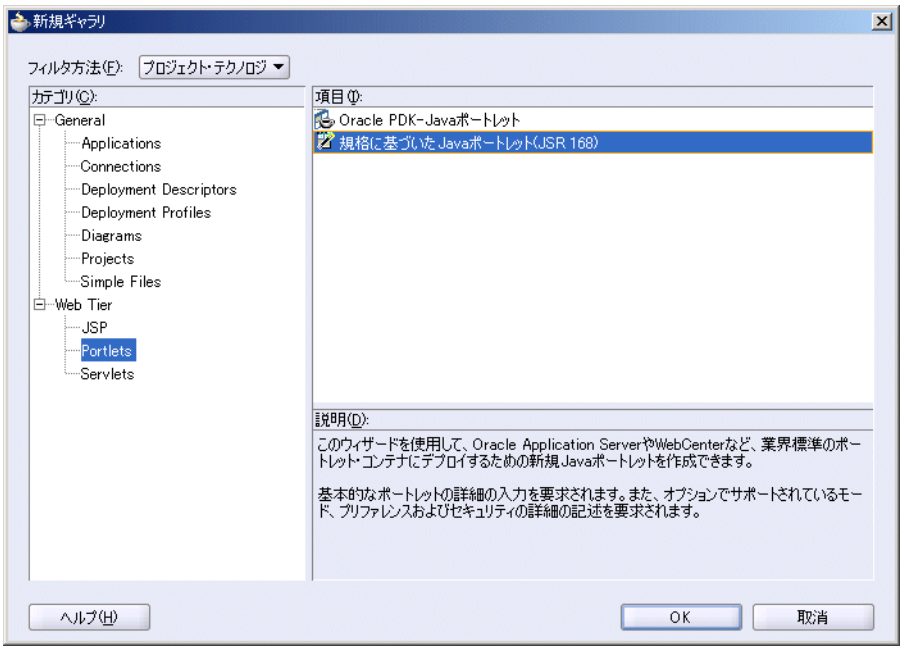
   - **Model:** ここでは、アプリケーションがバックエンド・ロジックを実行する場合に必要となる JavaBeans およびその他のデータ・コントロールを定義します。
   - **Portlets:** ここでは、ポートレットを作成します。
   - **ViewController:** ここでは、ポートレットを消費する JavaServer Faces ページを作成します。

図 3-4 新しい MySample アプリケーション

今度は、JSR 168 Java ポートレット・ウィザードを起動してポートレットを作成します。

10. アプリケーション・ナビゲータで、「**Portlets**」を右クリックし、「**新規**」を選択します。
11. 「カテゴリ」ペイン（図 3-5）で、「**Web Tier**」カテゴリを開き、「**Portlets**」をクリックします。

図 3-5 ポートレットの新規ギャラリ



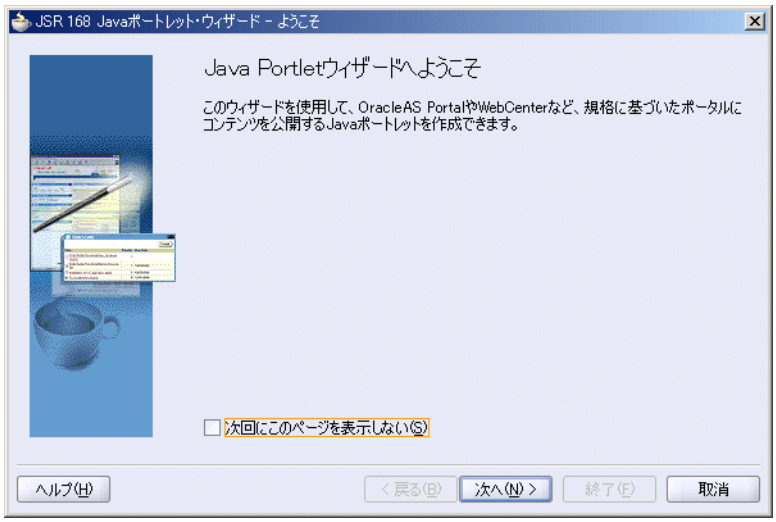
作成できるポートレットは、次の 2 種類です。

- Oracle PDK Java ポートレット。WebCenter アプリケーション、OracleAS Portal、またはその他のタイプの Oracle 固有ソリューションによって消費されるポートレットを構築する場合は、このオプションを選択します。Oracle PDK Java ポートレットは、PDK により提供された API を使用して構築します。
- 規格に基づいた（JSR 168）Java ポートレット。Java ポートレットは、ポートレット規格をサポートするあらゆるベンダーからのポータルによって消費できます。このチュートリアルでは、規格に基づいた（JSR 168）Java ポートレットを構築します。

12. **「規格に基づいた Java ポートレット (JSR 168)」**を選択し、**「OK」**をクリックします。

    これによって、JSR 168 Java ポートレット・ウィザードが開きます（図 3-6）。

図 3-6 JSR 168 Java ポートレット・ウィザードの「ようこそ」ページ

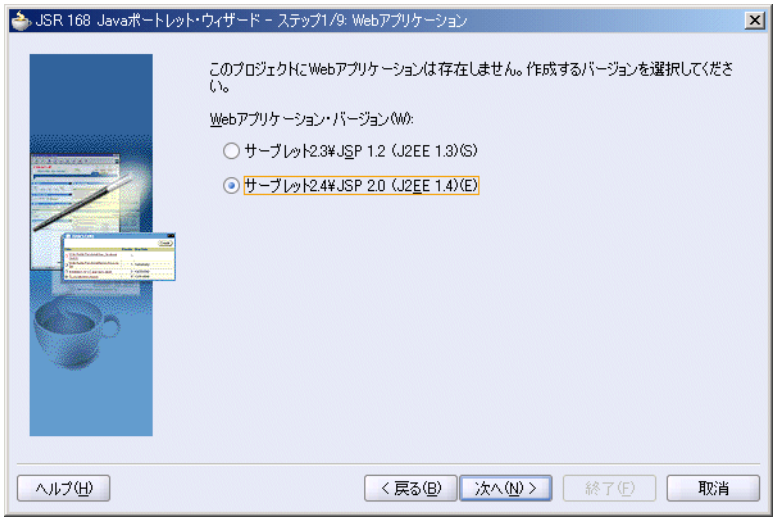


JSR 168 Java ポートレット・ウィザードによって、ポートレットのスケルトンが生成されます。このスケルトンに、独自のロジックを追加します。これから、その方法を示します。

13. **「次へ」**をクリックして、「ようこそ」ページの先に進みます。

14. 「Web アプリケーション」ページでは、指定されているデフォルトをそのままにしておきます（図 3-7）。サーブレット・バージョンの選択によって、使用可能なタグ・ライブラリが決まります。これらは下位互換性があるため、古いバージョンを選択するための特別な理由がないかぎりは、常に最新のバージョンを選択することをお薦めします。

図 3-7 Web アプリケーション・バージョンの選択

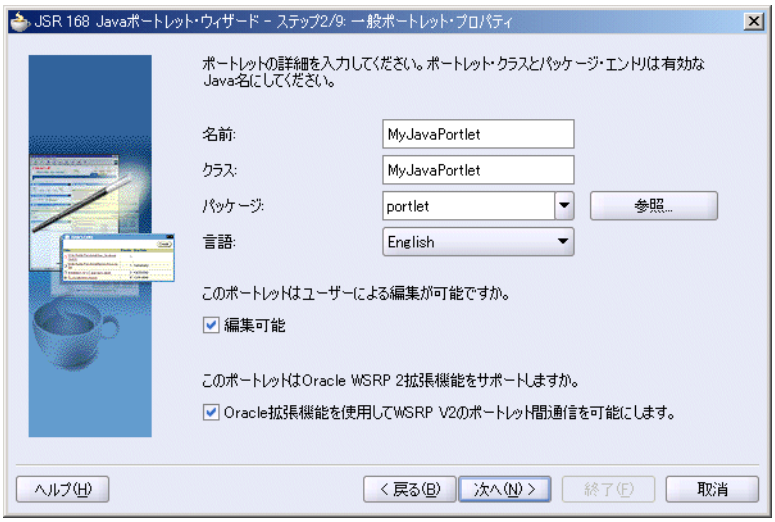


15. **「次へ」**をクリックします。

16. 「一般ポートレット・プロパティ」ページで、「名前」および「クラス」の名前を `MyJavaPortlet`（空白なし）に変更します（図 3-8 を参照）。

    通常は、ページに表示されるポートレットの名前とアプリケーション・ナビゲータに表示される名前が同じになるように、クラス名と表示名を同じにすることをお薦めします。

図 3-8「一般ポートレット・プロパティ」ページ



17. **「Oracle 拡張機能を使用して WSRP V2 のポートレット間通信を可能にします。」**を選択します。

    WSRP のポートレット間通信については、このチュートリアルの以降の部分で学びます。

18. 「一般ポートレット・プロパティ」ページに表示されているその他のデフォルトを受け入れます（図 3-8）。

    ユーザーが実行時にこのポートレットをパーソナライズできるように、**「編集可能」**ボックスが選択されていることを確認します。（「編集可能」を選択すると、ポートレットがパーソナライズ可能になります。ユーザーは、自分のみに適用する変更を行うとき、ポートレットをパーソナライズします。それより上位の権限セットを持つユーザーは、ポートレットをカスタマイズして、誰もに表示される変更を行うことができます。詳細は後で説明します。）

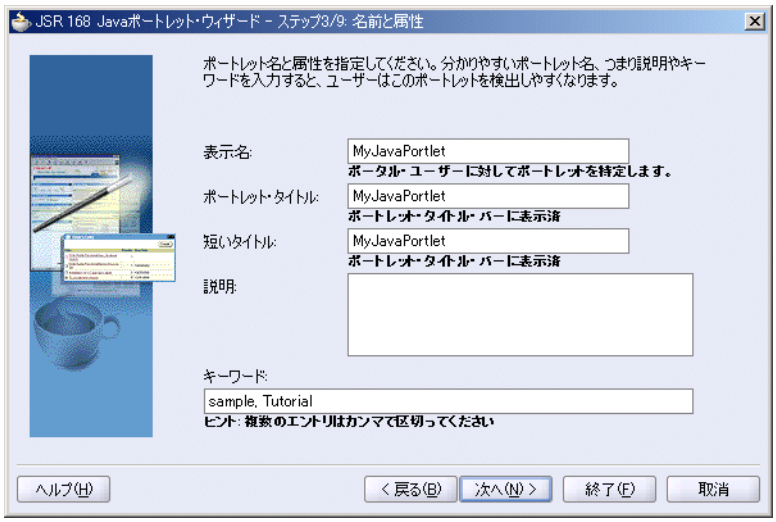
19. **「次へ」**をクリックします。

20. 「名前と属性」ページで、図 3-9 に示した値を入力します。

表 3-1「名前と属性」の値

プロパティ	値
表示名	JDeveloper のコンポーネント・パレットに表示される名前。クラス名として **MyJavaPortlet** を入力したため、このフィールドにはすでにその名前が移入されています。
ポートレット・タイトル	ポートレット・ヘッダーに表示されるタイトル。クラス名として **MyJavaPortlet** を入力したため、このフィールドにはすでにその名前が移入されています。
短いタイトル	モバイル機器のポートレット・ヘッダーに表示されるタイトル。このフィールドは使用しませんが、移入されたままにしておいてかまいません。
説明	ポートレットの説明。このフィールドは、ポートレットを OracleAS Portal 10*g* 環境で使用する場合にのみ関連します。このフィールドは空白のままにしておきます。
キーワード	**sample, Tutorial** と入力します。「キーワード」には、ユーザーが検索中にページ、項目またはポートレットを見つけられるように、これらの追加情報が表示されます。キーワードは Oracle WebCenter Suite でも OracleAS Portal 10*g* でもサポートされていませんが、他のベンダーではサポートされます（デプロイメント環境を他のベンダーから取得した場合）。

図 3-9「名前と属性」ページ



21. 「次へ」をクリックします。

「コンテンツ・タイプとポートレット・モード」ページで、text/html がデフォルトのコンテンツ・タイプになっていることに注意してください。これは、ポートレットが HTML でエンコードされたテキストをサポートするということです。text/html のデフォルトのポートレット・モードとして「view」と「edit」がリストされます。「view」はポートレット・モードとして常に使用可能です。「edit」モードを選択すると、ユーザーがポートレット・インスタンスをパーソナライズできるページが提供されます。

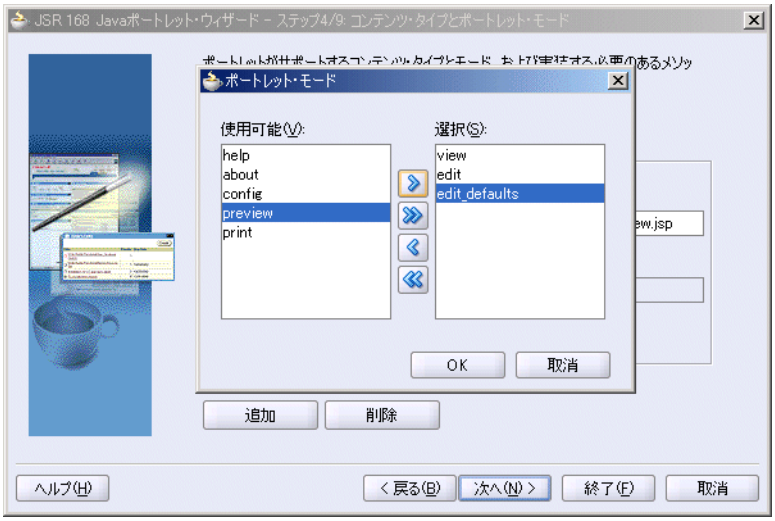
このチュートリアルの以降の部分で、このポートレットを使用したパーソナライズとカスタマイズの違いをテストします。このため、ここでポートレットのカスタマイズを有効にしておく必要があります。カスタマイズを有効にするには、「`edit_defaults`」コンテンツ・タイプをポートレットの構成に追加します。

22. 「コンテンツ・タイプとポートレット・モード」ペイン（図 3-10）で、**「view」**を選択し、**「追加」**をクリックします。

    ポートレット・モード・ウィンドウが表示されます。

23. **「edit_defaults」**を選択し、矢印を使用して**「選択」**ペインに移動します。

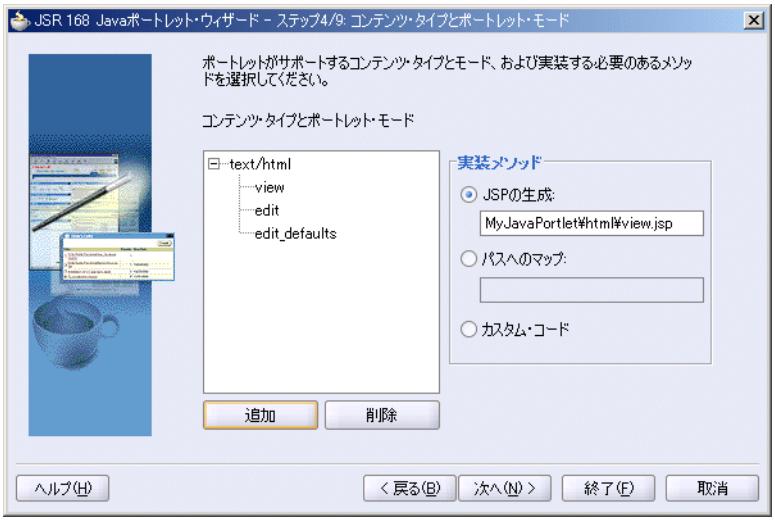
図 3-10 ポートレット・モードの追加



24. **「OK」**をクリックします。

    「コンテンツ・タイプとポートレット・モード」ページの先に進む前に、**「実装メソッド」**領域（図 3-11）に注目してください。これらのコントロールによって、ポートレットに JSP を生成するかどうか、あるいは独自のカスタム JSP コードを使用するかどうかを指定できます。このレッスンでは、JSP を生成するように JDeveloper を設定します。

図 3-11 実装メソッドの選択



25. **「次へ」**をクリックします。

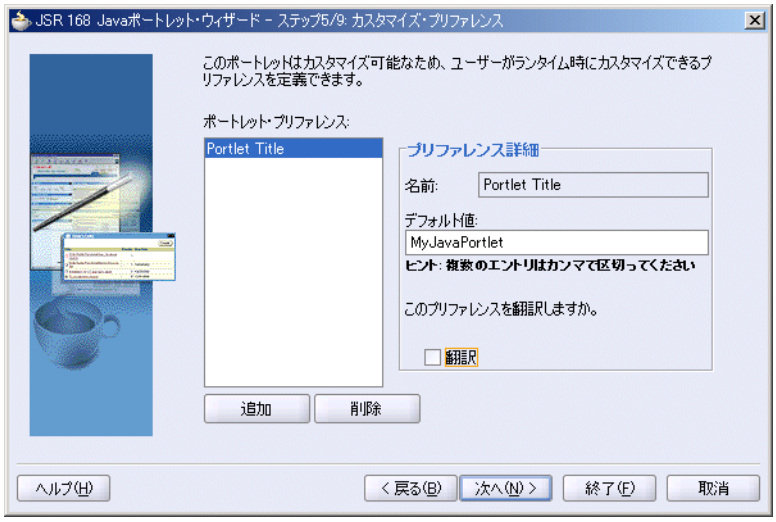
    この時点で**「終了」**をクリックして基本的なポートレットを生成することもできますが、ここでは、他の使用可能なオプションを調べるために引き続きウィザードを使用します。

26. 「カスタマイズ・プリファレンス」ページ（図 3-12）で、「デフォルト値」フィールドに MyJavaPortlet を入力します。

    このポートレットを**「編集可能」**にすることを指定したため、ユーザーは実行時にこのタイトルを変更できます。

    このリリースでは翻訳は関係ないため、**「翻訳」**チェック・ボックスの選択を解除します。

図 3-12 デフォルト値の設定



    ここではこのページは何も変更しませんが、将来的に、このページを使用して、ポートレットの他のカスタマイズ・オプションを追加できます。たとえば、ポートレットが Zip Code パラメータを受け入れる場合、ユーザーが Zip Code ラベルをパーソナライズできるようにします。この場合は、「追加」ボタンを使用して、Zip Code ラベルをパーソナライズ可能にします。

27. **「次へ」**をクリックします。

28. 「セキュリティ・ロール」ページで、**「次へ」**をクリックします。

    このページを使用して、このポートレットに対して設定するアプリケーションのセキュリティ・ロールを指定します。

29. 「キャッシュ」ページで、**「次へ」**をクリックします。

    このページの設定によって、ポートレットの有効期限に基づいたキャッシュを定義できます。ここではキャッシュ条件は必要ありません。

30. 「初期化パラメータ」ページで、**「次へ」**をクリックします。

    初期化パラメータは、Web アプリケーション開発者がポートレットの動作を構成する手段です。このチュートリアルでは、初期化パラメータは必要ありません。

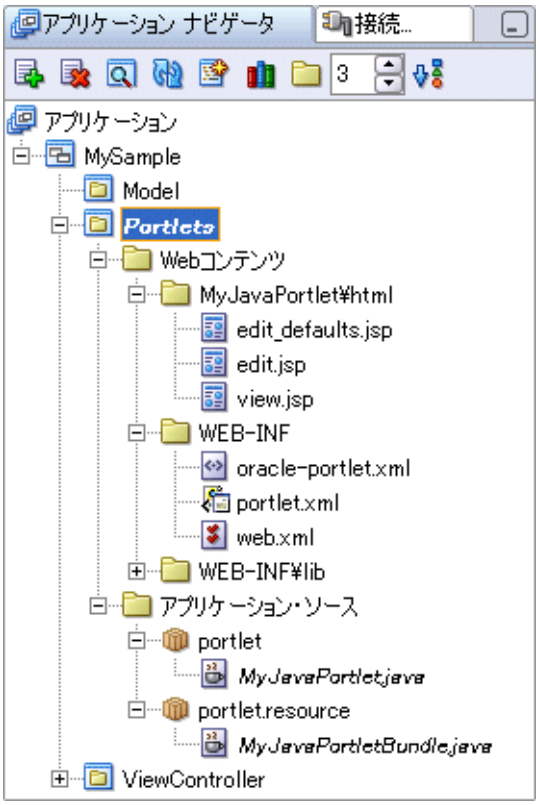
31. 「ポートレット・ナビゲーション・パラメータ」ページで、**「次へ」**をクリックします。

    ナビゲーション・パラメータは、WSRP 2.0 の機能です。このページでは、JSR 168 ポートレットによって消費される外部パラメータを指定できます。このチュートリアルでは、ナビゲーション・パラメータは必要ありません。

32. **「終了」**をクリックします。

「終了」をクリックした後、アプリケーション・ナビゲータの「ポートレット」プロジェクトの下に、新しく生成されたいくつかのファイルが表示されます。展開されたナビゲータは、図 3-13 のようになります。

図 3-13 新しいポートレットに対して生成されたファイル



- 「アプリケーション・ソース」の下の「portlet」および「portlets.resource」内に、次の 2 つの Java クラスがあります。
  - （「portlet」の下の）**MyJavaPortlet.java** は、ポートレット・コンテナによって起動されます。これには、ポートレット標準で必要なすべてのメソッドが含まれます。
  - （「portlet.resource」の下の）**MyJavaPortletBundle.java** には、ポートレットのすべての変換文字列が含まれます。
- 「Web コンテンツ」の下の「MyJavaPortlet¥html」内に、次のファイルがあります。
  - **edit_defaults.jsp:** これには、「カスタマイズ」ダイアログへの移入に必要な情報が含まれます。
  - **edit.jsp:** これには、「パーソナライズ」ダイアログへの移入に必要な情報が含まれます。
  - **view.jsp:** これは、ポートレットが他のコンポーネントとページを共有しているときに起動されます。

- 「Web コンテンツ」の下の「WEB-INF」内に、次の 3 つのデプロイメント・ディスクリプタがあります。
  - **oracle-portlet.xml:** これには、インポート / エクスポートおよびポートレット間通信用の Oracle 拡張機能をサポートするための情報が含まれます。このファイルは、ウィザードのステップ 2 で「Oracle 拡張機能を使用して WSRP V2 のポートレット間通信を可能にします。」を選択したために表示されます。
  - **portlet.xml:** これにより、すべてのポートレット・リソース（JSR 168 Java ポートレット・ウィザードで入力した情報）が指定されます。
  - **web.xml:** これにより、Web アプリケーション・リソースが指定されます。

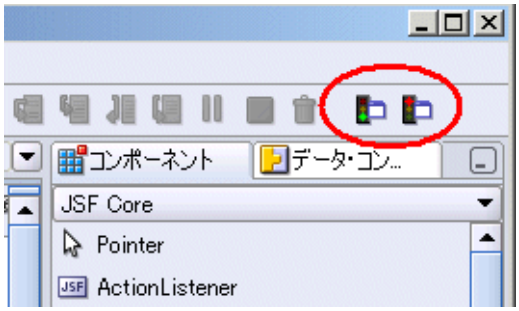
ポートレットの作成が完了したので、今度はポートレットを実行してその動作を確認します。これを行うには、まず、ポートレットを Oracle Application Server にデプロイする必要があります。JDeveloper をインストールしたときに、WebCenter Preconfigured OC4J というスタンドアロン OC4J が自動的にダウンロードされているため、それを使用できます。ただし、最初に、この OC4J と使用している JDeveloper のインスタンスとの間の接続を確立する必要があります。これから、それを行います。

手順 2: 接続の設定

ポートレットをデプロイする場合、使用しているいずれかのアプリケーション・サーバーへの接続を確立する必要があります。このチュートリアルでは、ポートレットを WebCenter Preconfigured OC4J にデプロイします。これから、この OC4J への接続を設定します。

1. JDeveloper ツールバーの一番右にある**「WebCenter Preconfigured OC4J の起動」**アイコン（図 3-14）をクリックして、WebCenter Preconfigured OC4J を起動します。

図 3-14 Preconfigured OC4J の起動

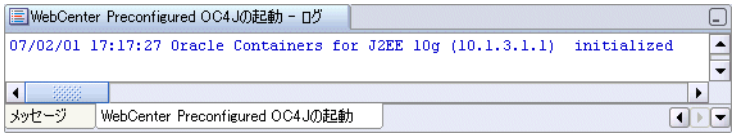


初めてこの OC4J を起動した場合は、WebCenter Preconfigured OC4J をインストールするかどうか尋ねるメッセージが表示されます。

2. **「はい」**をクリックします。

OC4J インスタンスの初期化が完了したことを示すメッセージを待ってから、次に進みます（図 3-15 を参照）。

図 3-15 OC4J 初期化メッセージ



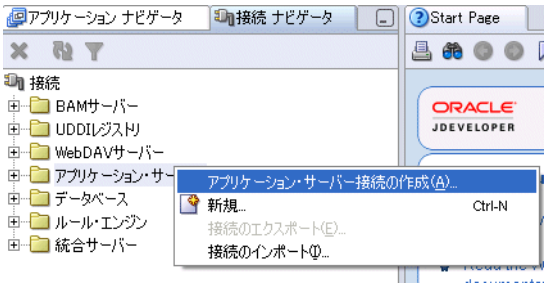
Preconfigured OC4J を初めて起動すると、JDeveloper エディタに README ファイルが表示されます。このドキュメントには、Preconfigured OC4J の起動、停止および接続の方法が記載されています。また、ポートのデフォルトがリストされ、有用なトラブルシュー

ティング・ヒントが示されています。将来このファイルにアクセスする必要が生じた場合は、JDeveloper のメイン・メニューから**「ヘルプ」**、**「WebCenter Preconfigured OC4J README」**の順に選択します。

Preconfigured OC4J はコンピュータにインストールされましたが、さらに（JDeveloper が Preconfigured OC4J の場所を認識できるように）Preconfigured OC4J への接続を確立する必要があります。

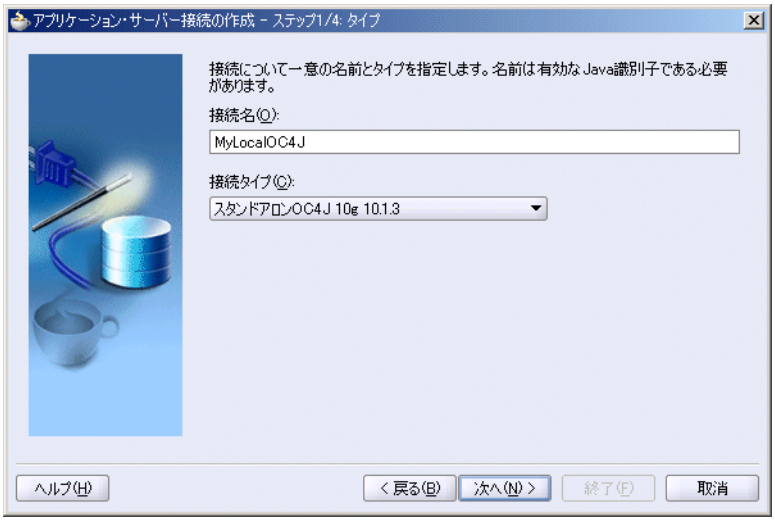
3. （「アプリケーション・ナビゲータ」の隣の）**「接続ナビゲータ」**をクリックします。
4. **「接続」**ノードを右クリックし、**「アプリケーション・サーバー接続の作成」**を選択します（図 3-16 を参照）。

図 3-16 JDeveloper と Preconfigured OC4J の接続



5. **「次へ」**をクリックして、「ようこそ」ページを終了します。
6. 「接続名」フィールドに `MyLocalOC4J` を入力します。
7. 「接続タイプ」リストから、**「スタンドアロン OC4J 10g 10.1.3」**を選択します（図 3-17 を参照）。

図 3-17 接続名および接続タイプの選択

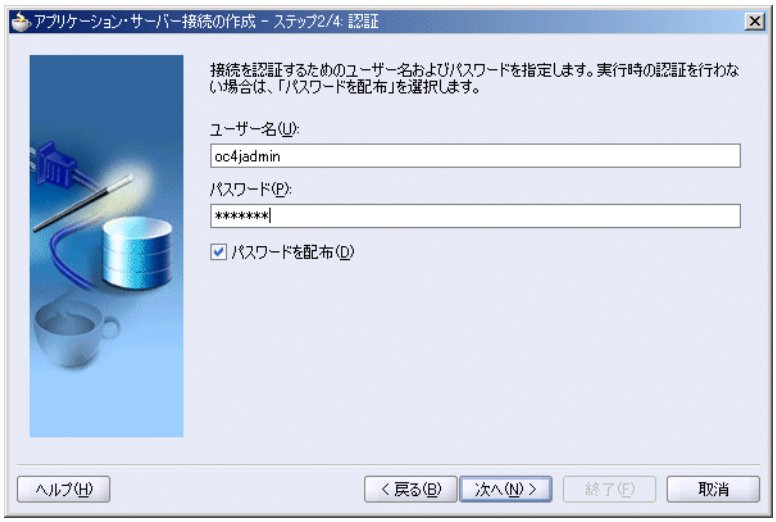


8. **「次へ」**をクリックします。
9. ユーザー名として `oc4jadmin` を入力し、パスワードとして `welcome` を入力します。

   `welcome` は、WebCenter Preconfigured OC4J のデフォルト・パスワードです。

10. 接続を確立するたびにパスワードを入力しなくてもよいように、**「パスワードを配布」**を選択します（図 3-18）。

図 3-18 ログイン資格証明の入力



11. **「次へ」**をクリックします。

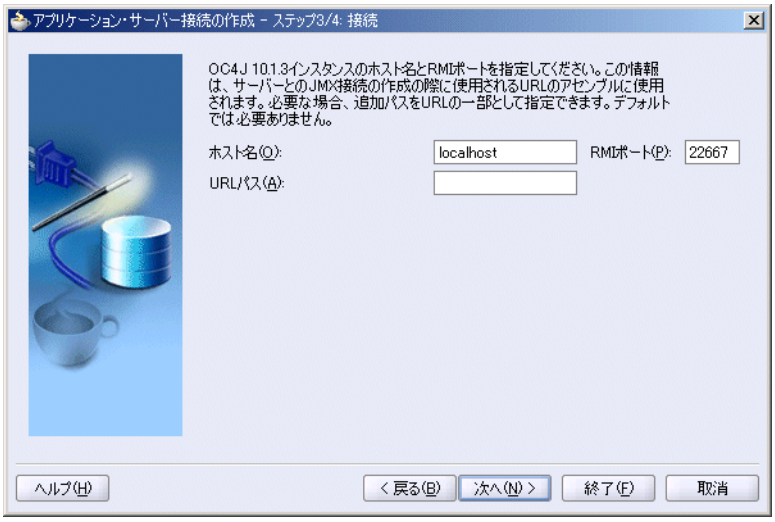
12. 使用しているコンピュータに WebCenter Preconfigured OC4J が存在している場合は、「ホスト名」でデフォルトの「`localhost`」を受け入れます。

    このチュートリアルでは、URL 内に `localhost` を使用するとローカル・コンピュータを参照できることを前提としています。ただし、ファイアウォール構成によってはこのアドレスへのアクセスがブロックされることもあります。このような場合は、`localhost` が許可されるようにファイアウォール構成を変更するか、ここにコンピュータの固定 IP アドレスを入力します。使用しているコンピュータが DHCP を使用するように構成されていて固定 IP アドレスを持たない場合は、`localhost` が機能するようにファイアウォールを再構成してください。

13. 「RMI ポート」を 22667（Preconfigured OC4J のデフォルトの RMI ポート）に変更します。

    「URL パス」は空白のままでかまいません。この情報は必要ありません（図 3-19）。

図 3-19 Preconfigured OC4J の接続詳細の入力



14. **「次へ」**をクリックします。

15. **「接続のテスト」**をクリックして、接続を確認します。

    何も問題がなければ、ステータス・ペインに成功メッセージが表示されます。テストが失敗すると、ステータス・フィールドにエラー・メッセージが表示されます。ウィザードの前のページに戻ってエントリを訂正するには、「戻る」ボタンを使用します。

16. **「終了」**をクリックします。

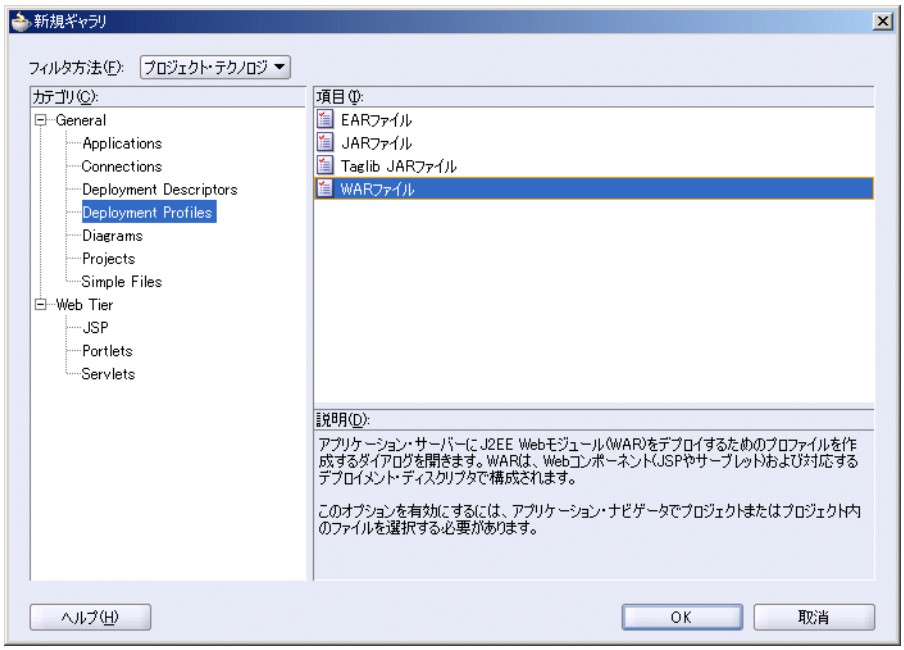
接続が確立されたので、ポートレットを WAR ファイルに含める準備ができました。WAR は web application archive の略語で、ポートレットのデプロイに必要なすべてのリソース、ポートレットおよびデプロイメント・ディスクリプタをまとめてパッケージ化したものです。

手順 3: ポートレットのデプロイ

このレッスンでは、Java ポートレットをローカルの WebCenter Preconfigured OC4J にデプロイする方法を学びます。ポートレットをデプロイする場合、J2EE サーバー上で実行できるようにポートレットをパッケージ化します。OracleAS Portal の知識がある場合、これは実際にはポートレット・プロバイダを作成することです。これは、WSRP の世界ではポートレット・プロデューサと呼ばれています。

1. **アプリケーション・ナビゲータ**を表示します。
2. **「Portlets」**プロジェクトを右クリックし、**「新規」**をクリックします。
3. 「カテゴリ」セクションで**「General」**を開き、**「Deployment Profiles」**をクリックします。
4. 「項目」セクションで**「WAR ファイル」**を選択します（図 3-20）。

図 3-20 WAR ファイルの作成



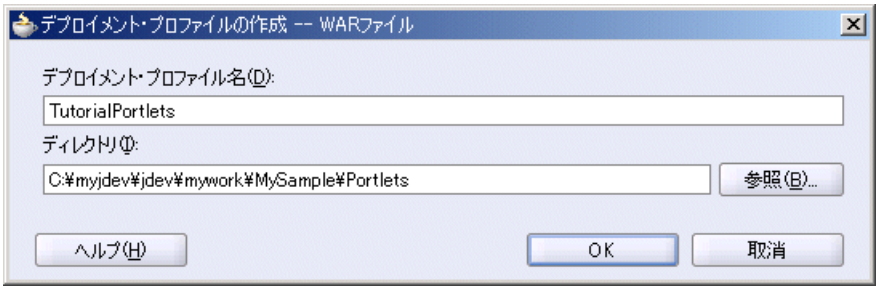
5. **「OK」**をクリックします。

6. 「デプロイメント・プロファイルの作成 -- WAR ファイル」ダイアログ・ボックス（図 3-21）で、次のように入力します。

表 3-2 デプロイメント・プロファイルの作成 -- WAR ファイル

設定	値
デプロイメント・プロファイル名	`TutorialPortlets` を入力します。
ディレクトリ	「手順 1: JSR 168 Java ポートレット・ウィザードの使用」の手順 7 で選択したディレクトリにナビゲートします。デフォルトを受け入れるだけでよいはずです。

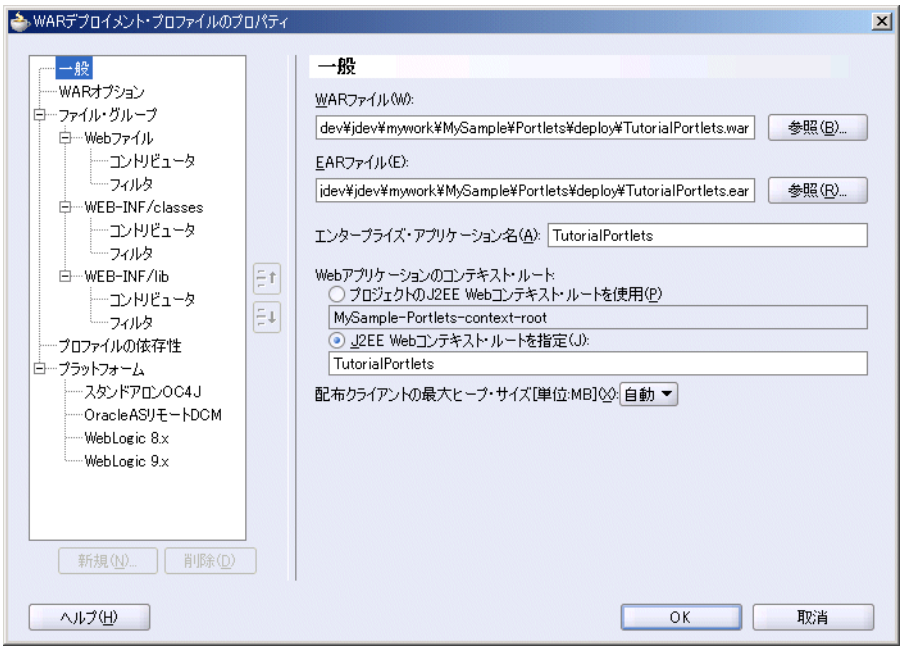
図 3-21 デプロイメント・プロファイルの作成 - WAR ファイル



7. 「**OK**」をクリックします。

8. WAR デプロイメント・プロファイルのプロパティ・ウィンドウで、「**J2EE Web コンテキスト・ルートを指定**」を選択し、`TutorialPortlets` を入力します（図 3-22 を参照）。

図 3-22 WAR デプロイメント・プロファイルのプロパティの設定

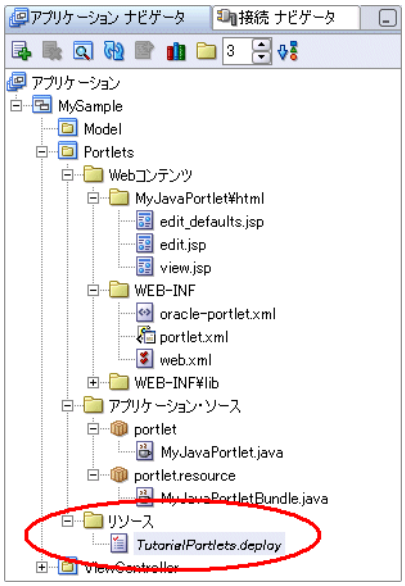


9. 「**OK**」をクリックします。

10. アプリケーション・ナビゲータで、**「リソース」**ノードを開きます。

    デプロイメント・プロファイル TutorialPortlets.deploy が表示されます（図 3-23 を参照）。

図 3-23 TutorialPortlets.deploy ファイル



11. **「TutorialPortlets.deploy」**を右クリックし、**「MyLocalOC4J に配布」**（MyLocalOC4J は先に作成した接続）をクリックします（「手順 2: 接続の設定」を参照）。

12. デプロイメント中にアプリケーションの構成ウィンドウが表示されたら、**「OK」**をクリックします。

    （JDeveloper ウィンドウの一番下にある）デプロイメント・ログに「デプロイが終了」というメッセージが表示されるのを待って、デプロイメントが正常に完了したことを確認します。

    今度はポートレットを実行して、適切に機能することを確認します。

13. ブラウザ・ウィンドウを起動し、次の URL を入力します。

    ```
    http://<host>:<port>/<context-root>/portlets/wsrp2?WSDL
    ```

    各パラメータの意味は次のとおりです。

パラメータ	値
host	WebCenter Preconfigured OC4J への接続に使用したホスト名。このレッスンで以前に、デフォルト `localhost` を受け入れました。 （localhost ではなくコンピュータの IP アドレスを使用するように選択した場合、ここにはその IP アドレスを入力してください。）
port	HTTP リスナー・ポート。WebCenter Preconfigured OC4J の場合は、この Preconfigured OC4J がリスニングするデフォルト・ポートである 6688 を使用してください。[1]
context-root	このレッスンの手順 8 で設定した TutorialPortlets。

[1] Preconfigured OC4J がリスニングするポートを変更するには、*JDEVHOME*¥jdev¥extensions¥oracle.adfp.seededoc4j.10.1.3.2.0¥j2ee¥home¥config¥default-web-site.xml に移動し、エントリ <web-site .... port="6688" ...> を変更します。

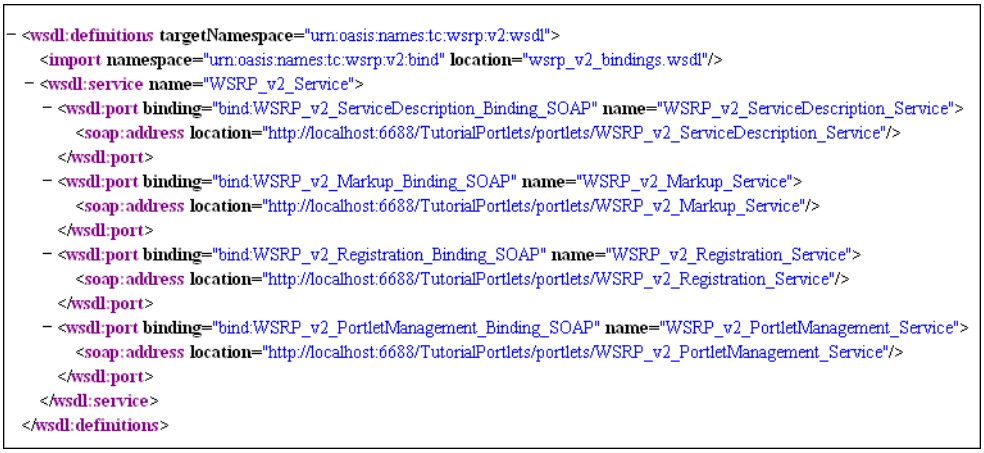
したがって、このチュートリアルでは、URL は次のようになります。

```
http://localhost:6688/TutorialPortlets/portlets/wsrp2?WSDL
```

この URL を書き留めておきます。この URL は、「手順 5: Oracle WebCenter Framework へのポートレットの登録」で必要になります。

14. ブラウザには、図 3-24 に示すような XML が表示されます。

図 3-24 Web サービスとしてのポートレットを記述する WSDL



デプロイしたポートレットが、Web サービスとして公開されています。ブラウザに表示されているのは、この Web サービスを記述する Web Services Description Language（WSDL）です。WSDL が図 3-24 のように表示されていれば、次の手順は、ポートレットを消費する JSF ページを作成することです。

手順 4: JSF ページの作成

JavaServer Faces（JSF）ページを作成するには、次のようにします。

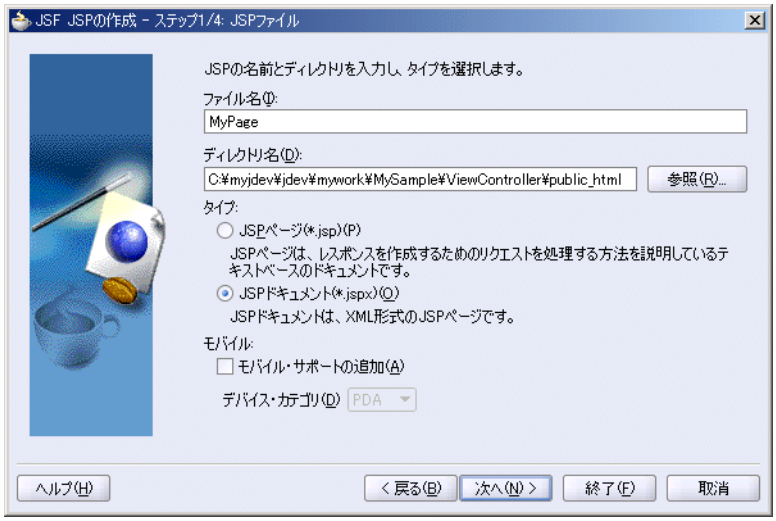
1. アプリケーション・ナビゲータで、**「ViewController」**プロジェクトを右クリックし、**「新規」**を選択します。
2. 「カテゴリ」ペインで、「Web Tier」の下の**「JSF」**をクリックして、JavaServer Faces ページを作成します。
3. 「項目」で**「JSF JSP」**を選択し、**「OK」**をクリックします。

   JSF JSP ウィザードが開きます。
4. **「次へ」**をクリックして、「ようこそ」ページを終了します。
5. 「ファイル名」フィールド（図 3-25）に `MyPage` を入力します。

   「ディレクトリ名」には、デフォルト場所を受け入れてかまいません。

6. 「タイプ」で、ページの XML バージョン（つまり、.jspx ファイル）が作成されるように、**「JSP ドキュメント」**をクリックします。

図 3-25 JSP ページの作成



7. **「次へ」**をクリックします。

8. （少なくともこのレッスンでは）このページにはバックエンド・ロジックを追加しないため、新しいマネージド Bean は必要ありません。デフォルト設定（**マネージド Bean で UI コンポーネントを自動公開しない**）のままにしておきます（図 3-26 を参照）。

図 3-26 JSF JSP の作成ウィザードの「コンポーネント・バインディング」ページ

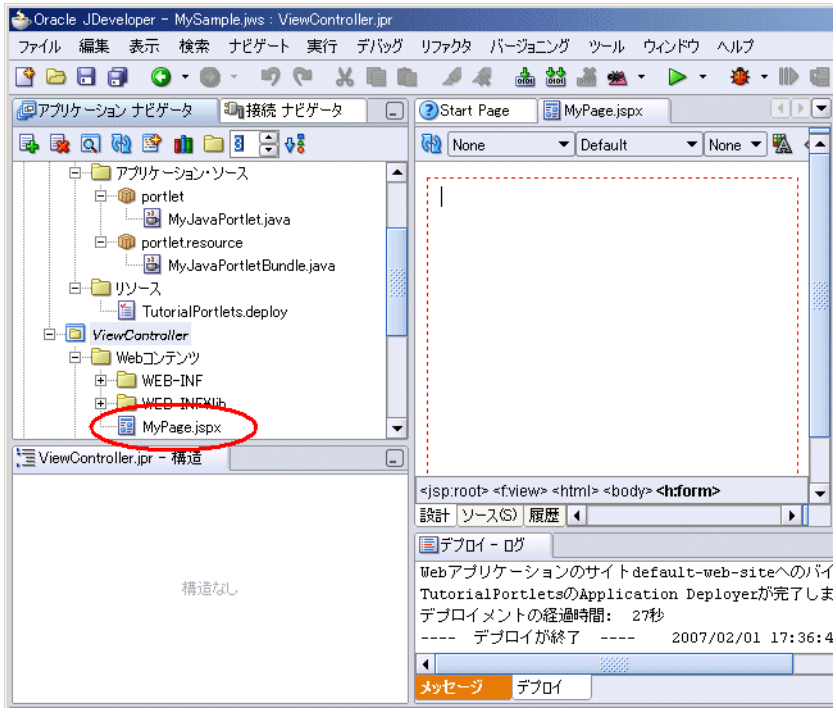
9. **「次へ」**をクリックします。

10. 「タグ・ライブラリ」ページで、二重矢印を使用して、「使用可能なライブラリ」ペインから「選択済のライブラリ」ペインにすべてのライブラリを移動します。次のライブラリを含める必要があります。
    - ADF Faces Components 10_1_3_2_0
    - ADF Faces HTML 10_1_3_2_0
    - ADF Portlet Components 10_1_3_2_0
    - Customizable Components Core 10_1_3_2
    - JSF Core 1.0
    - JSF HTML 1.0
11. **「次へ」**をクリックします。
12. 特別な HTML オプションは必要ないため、図 3-27 に示すページで**「終了」**をクリックします。

図 3-27 JSF JSP の作成 - HTML オプション

アプリケーション・ナビゲータで、「ViewController」の「Web コンテンツ」の下に、作成したページ MyPage.jspx が表示されます（図 3-28 を参照）。

図 3-28 アプリケーション・ナビゲータでの MyPage



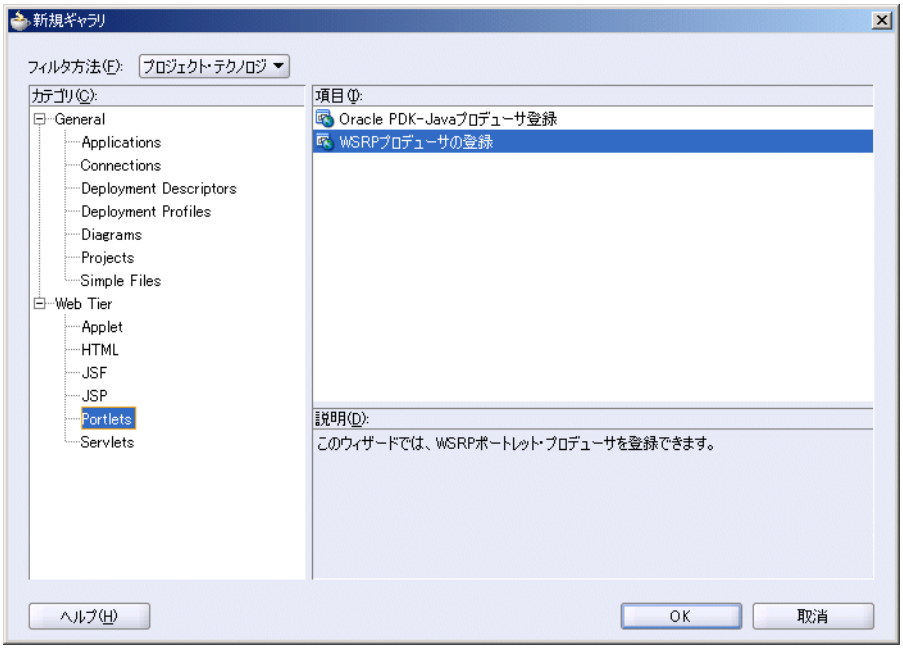
また、ビジュアル・エディタにページが開かれ、コンポーネントをすぐに追加できるようになります。

手順 5: Oracle WebCenter Framework へのポートレットの登録

これまでに、ポートレット、ポートレット・プロデューサおよび JavaServer Faces ページを作成しました。今度は、これらをすべて結合し、プロデューサへのアクセス方法をページに指示する必要があります。これは、アプリケーションへのプロデューサの登録とも呼ばれます。

1. アプリケーション・ナビゲータで、**「ViewController」**を右クリックし、**「新規」**をクリックします。
2. 新規ギャラリで、「Web Tier」の下の**「Portlets」**をクリックします（図 3-29 を参照）。

図 3-29 新しい WSRP プロデューサの登録



3. **「WSRP プロデューサの登録」**を選択し、**「OK」**をクリックします。

   これによって、WSRP プロデューサの登録ウィザードが開きます。
4. **「次へ」**をクリックして、「ようこそ」ページを終了します。
5. 「名前」フィールドに `TutorialProducer` を入力します。
6. **「次へ」**をクリックします。

7. 「接続」ページ（図 3-30）で、「手順 3: ポートレットのデプロイ」（サブ手順 13）で構成した URL を入力します。次に例を示します。

   ```
   http://localhost:6688/TutorialPortlets/portlets/wsrp2?WSDL
   ```

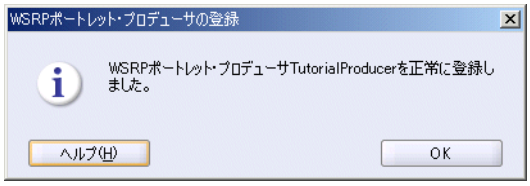
図 3-30 新しい WSRP プロデューサの接続詳細

   ローカルの Preconfigured OC4J を使用しているため、このチュートリアルではプロキシ情報は必要ありません。

8. **「次へ」**をクリックします。
9. 再度**「次へ」**をクリックして、デフォルトのタイムアウト値の 30 秒を受け入れます。
10. **「次へ」**をクリックして、デフォルトのセキュリティ設定を受け入れます。
11. 「キーストア」ページで、**「終了」**をクリックします。

    図 3-31 のようなメッセージが表示されます。

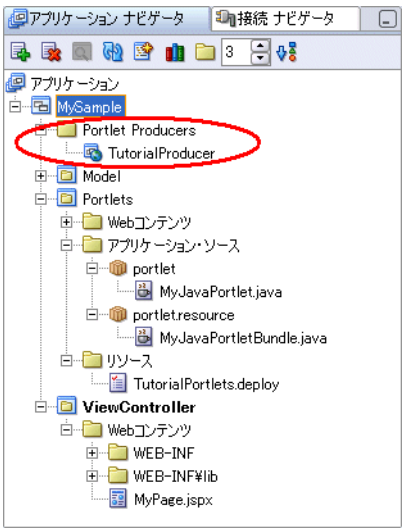
図 3-31 正常に登録されたポートレット・プロデューサ



12. **「OK」**をクリックして、このメッセージ・ボックスを閉じます。

アプリケーション・ナビゲータに、「Portlet Producers」ノードが表示されます（図 3-32 を参照）。このノードを開き、TutorialProducer が表示されることを確認します。

図 3-32 アプリケーション・ナビゲータでの TutorialProducer



これで、ポートレット MyJavaPortlet の位置とそのアクセス方法がページに認識されます。これから、このことを確認します。

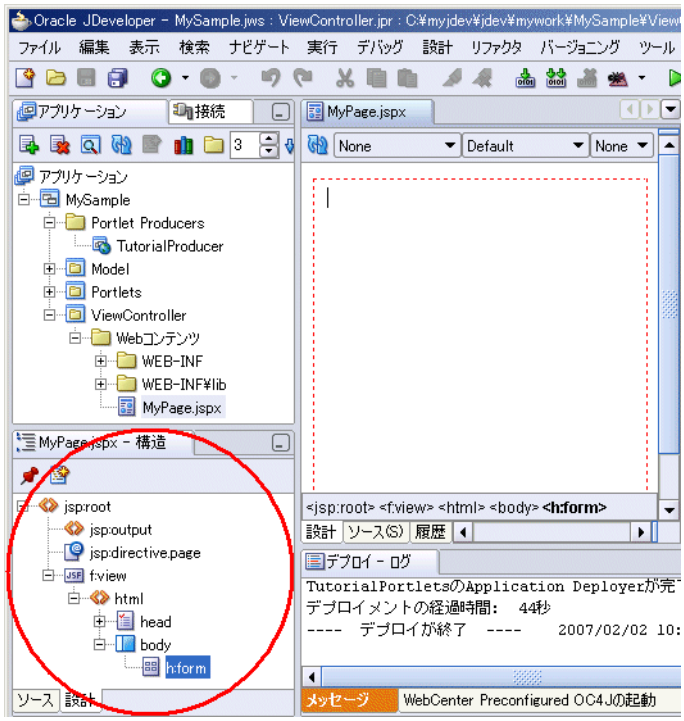
手順 6: ポートレットのテスト

ポートレットをテストするには、ポートレットを MyPage.jspx に追加してページを実行し、ポートレットが予測どおりに表示されるかどうかを確認します。

1. MyPage.jspx がまだ開いていない場合は、アプリケーション・ナビゲータで（「ViewController」の「Web コンテンツ」の下の）ページ名を見つけてダブルクリックします。

   これによって、構造ウィンドウ内にページが開きます（図 3-33 を参照）。

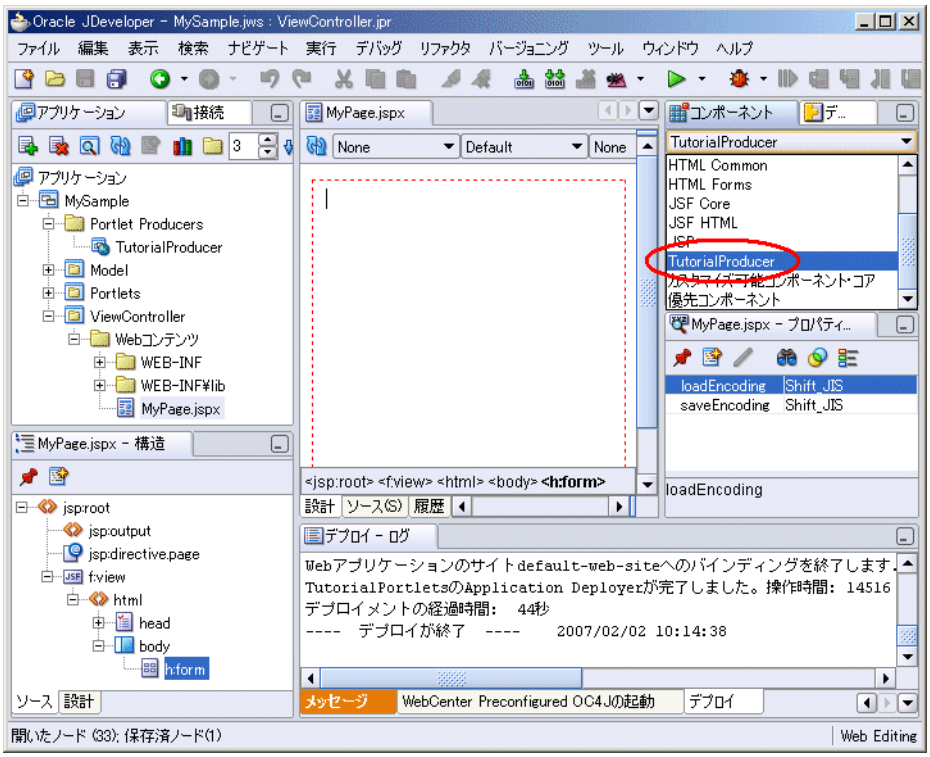
図 3-33 構造ウィンドウ内に開かれた MyPage.jspx



2. JDeveloper ウィンドウの一番右にあるコンポーネント・パレットのドロップダウン・リスト（図 3-34）をクリックし、TutorialProducer がここに表示されていることを確認します。

> **注意：** コンポーネント・パレットが表示されていないこともあります。表示されていない場合は、JDeveloper メニューから「表示」、「コンポーネント・パレット」の順に選択してください。

図 3-34 コンポーネント・パレットでの TutorialProducer

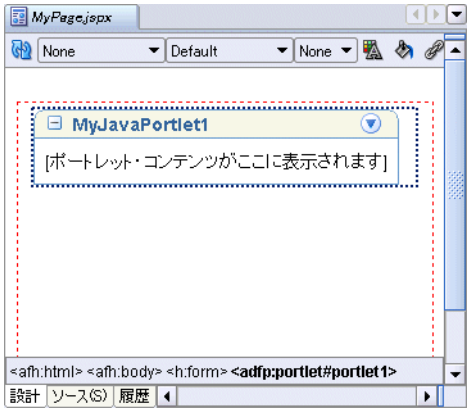


3. 「**TutorialProducer**」を選択します。ポートレット MyJavaPortlet がこの場所に表示されます。

4. 「**MyJavaPortlet**」を選択し、構造ウィンドウ内の最後のエントリ「`h:form`」にドラッグします。ポートレットがフォームに含まれていることを確認できます。構造ウィンドウで、これを簡単に確認できます。

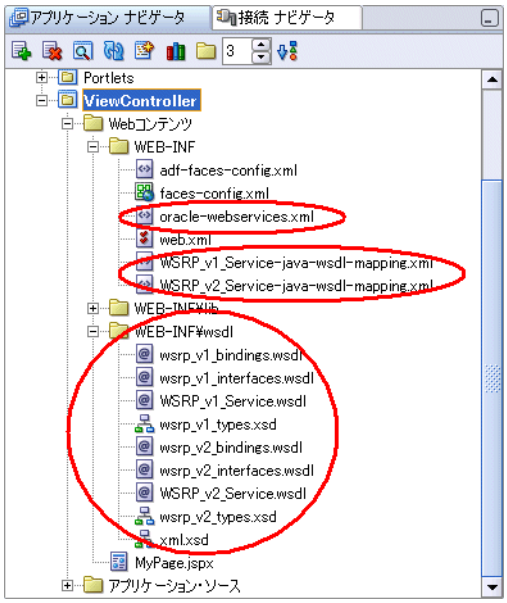
現在、図 3-35 のように表示されています。

図 3-35 MyPage.jspx に追加された MyJavaPortlet



新しく生成されたいくつかのファイルおよびフォルダは、アプリケーション・ナビゲータで表示されます。展開されたナビゲータは、図 3-36 のようになります。WSRP プロデューサのポートレットを消費するときに、これらの内部ファイルが作成されます。これらのファイルはいずれも編集する必要がありません。

図 3-36 アプリケーション・ナビゲータ内に作成された WEB-INF¥wsdl



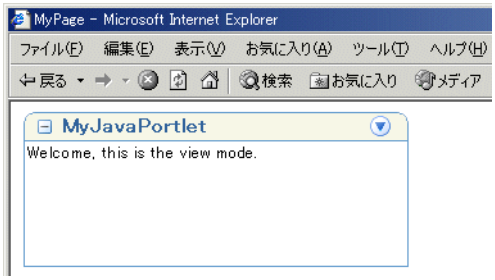
今度は、ページを実行します。

5. アプリケーション・ナビゲータで、**「MyPage.jspx」**を右クリックし、**「実行」**を選択します。

   これには数秒かかることがあります。「実行中 : 埋込み OC4J サーバー」というメッセージ・ログが表示されます。ポートレットはこのチュートリアルの最初にダウンロードした Preconfigured OC4J 内で実行されていますが、ページそのものは JDeveloper の埋込み OC4J 内で実行されています。

   ページが新しいブラウザ・ウィンドウ内に開きます（図 3-37）。

図 3-37 ブラウザ・ウィンドウ内の MyJavaPortlet

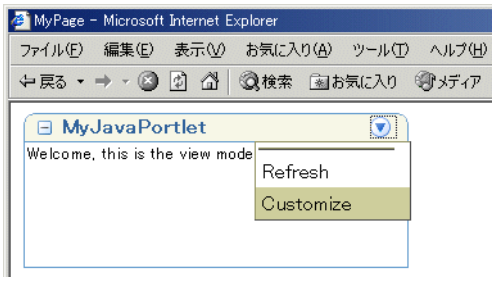


6. ポートレット・ヘッダーの**アクション・**メニューをクリックします。

   2 つのオプション（「リフレッシュ」および「カスタマイズ」）が表示されます（図 3-38）。「パーソナライズ」オプションは表示されないことに注意してください。MyJavaPortlet ではユーザーのパーソナライズをサポートしていますが、「パーソナライズ」オプションが表示されるのは、セキュリティ・モデルを実装するアプリケーション内にポートレットが表示されていて、かつ有効なユーザー資格証明でログインしている場合に限られます。「パーソナライズ」モードは、第 8 章「セキュリティの設定」でテストします。

   ここでは、「カスタマイズ」モードをテストします。

図 3-38 MyJavaPortlet のアクション



7. アクション・ドロップダウン・メニューから、**「カスタマイズ」**を選択します。

8. 「ポートレット・タイトル」フィールドで、ポートレット・ヘッダーを他の名前（たとえば、My First Java Portlet）に変更します（図 3-39 を参照）。

図 3-39 ポートレット・タイトルの変更

9. 「**OK**」をクリックします。

   ポートレットがブラウザに再表示され、選択した名前がヘッダーに表示されます。

   さて、このポートレットは適切に機能していることがわかりましたが、あまり面白みはありません。今度は、レンダリングされるポートレットのテキスト・ボックスにユーザーが HTML を貼り付けることができるように、いくつかのロジックを追加します。

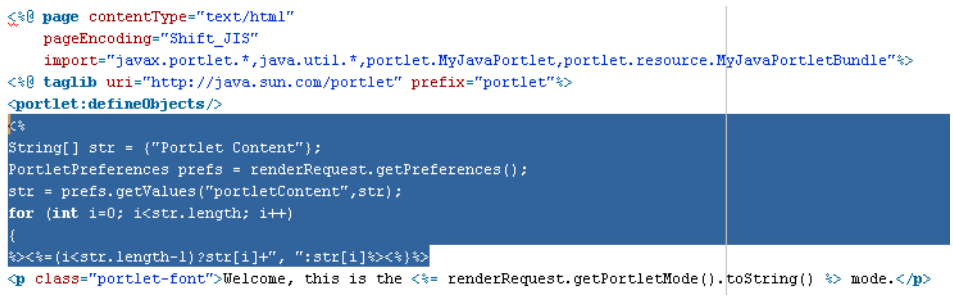
手順 7: ポートレットへの簡単なロジックの追加

この手順では、レンダリングされるポートレットのテキスト・ボックスにユーザーが HTML を貼り付けることを可能にするコードを追加します。

1. JDeveloper に戻ります。
2. アプリケーション・ナビゲータで、「Portlets」、「Web コンテンツ」、「MyJavaPortlet¥html」の下の「**view.jsp**」をダブルクリックします。

   アプリケーション・ナビゲータの右側に、新しい編集ウィンドウが開きます。
3. 新しいウィンドウの一番下にある「**ソース**」タブをクリックします。
4. 既存の JSP コードに、図 3-40 で強調表示されている行を追加します。

図 3-40 view.jsp の編集



コピーして貼り付けるコードは、次のとおりです。

```
<%
String[] str = {"Portlet Content"};
PortletPreferences prefs = renderRequest.getPreferences();
str = prefs.getValues("portletContent",str);
for (int i=0; i<str.length; i++)
{
%><%=(i<str.length-1)?str[i]+", ":str[i]%><%}%>
```

このコードによって、「パーソナライズ」モードまたは「カスタマイズ」モードで設定セットが取得され、ポートレットの「表示」モードでユーザーに表示されます。

5. JDeveloper のツールバーで、**「保存」**アイコンをクリックします。
6. **「edit.jsp」**をダブルクリックして、ビジュアル・エディタでこれを開きます。
7. **「ソース」**タブをクリックします。
8. 図 3-41 に示されたコードを追加して、「コンテンツ」という名前のフォーム・フィールドを実装します。コピーして貼り付けるコードは、次のとおりです。

```
<%
    String[] str = {"Portlet Content"};
    str = prefs.getValues("portletContent",str);
%>
<tr><td width="20%">
  <p class="portlet-form-field" align="right"> Content</p>
</td><td width="80%">
 <textarea rows="10" cols="60" class="portlet-form-input-field"
name="portletContent"><%
 for (int i=0; i<str.length; i++)
 {%><%= (i<str.length-1) ? str[i]+", " : str[i] %><%}%>
</textarea>
</td></tr>
```

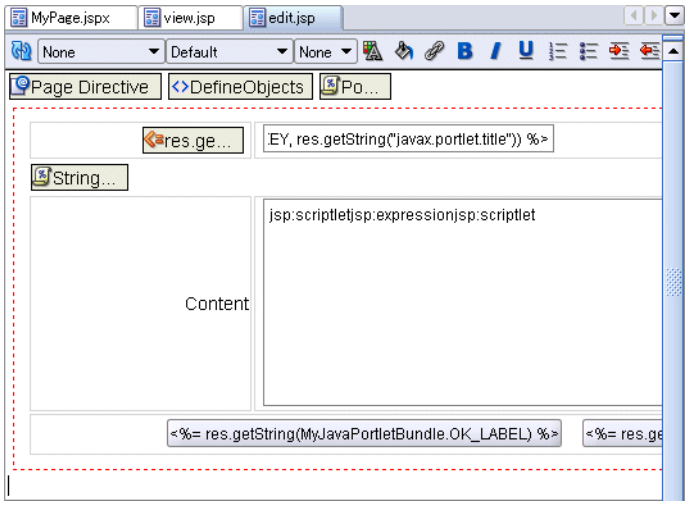
図 3-41 edit.jsp へのコードの追加

```
<FORM ACTION="<portlet:actionURL/>" METHOD="POST">
<TABLE BORDER="0">
<TR><TD WIDTH="20%">
<P CLASS="portlet-form-field" ALIGN="right">
<%= res.getString(MyJavaPortletBundle.PORTLETTITLE) %>
</P></TD><TD WIDTH="80%">
<INPUT CLASS="portlet-form-input-field" TYPE="TEXT"
  NAME="<%= MyJavaPortlet.PORTLETTITLE_KEY %>"
  VALUE="<%= prefs.getValue(MyJavaPortlet.PORTLETTITLE_KEY, res.getString("javax.portlet.title")) %>"
  SIZE="20">
</TD></TR>
<% String[] str = {"Portlet Content"};
   str = prefs.getValues("portletContent",str); %>
<tr><td width="20%">
 <p class="portlet-form-field" align="right"> Content</p>
</td><td width="80%">
 <textarea rows="10" cols="60" class="portlet-form-input-field" name="portletContent"><%
    for (int i=0; i<str.length; i++)
    {%><%= (i<str.length-1) ? str[i]+", " : str[i] %><%}%>
 </textarea>
</td></tr>
<TR><TD COLSPAN="2" ALIGN="CENTER">
<INPUT CLASS="portlet-form-button" TYPE="SUBMIT" NAME="<%= MyJavaPortlet.OK_ACTION %>"
  VALUE="<%= res.getString(MyJavaPortletBundle.OK_LABEL) %>">
<INPUT CLASS="portlet-form-button" TYPE="SUBMIT" NAME="<%= MyJavaPortlet.APPLY_ACTION %>"
  VALUE="<%= res.getString(MyJavaPortletBundle.APPLY_LABEL) %>">
</TD></TR>
</TABLE>
</FORM>
```

9. JDeveloper のツールバーで、**「保存」**アイコンをクリックします。

10. **「設計」**タブをクリックして、追加したフォーム・フィールドを確認します（図 3-42 を参照）。

図 3-42「コンテンツ」フィールドの追加



これで、「パーソナライズ」モードで変更した内容を格納するファイルを編集しました。今度は、「カスタマイズ」モードで変更した内容を格納するファイル（edit_defaults.jsp）にも同様の変更を行う必要があります。

11. **「edit_defaults.jsp」**をダブルクリックし、同じ場所に同じコードを追加します。完了後、必ず**「保存」**アイコンをクリックしてください。

    MyJavaPortlet.java（このポートレットの Java コードが含まれるファイル）も、同様に編集する必要があります。

12. アプリケーション・ナビゲータで、「Portlets」、「アプリケーション・ソース」、「portlet」の下の**「MyJavaPortlet.java」**をダブルクリックして、ビジュアル・エディタの「ソース」ビューでこれを開きます。

13. 「processAction」メソッドにスクロールダウンし、`//Save the preferences` というコード行を見つけます。次の 2 行の（太字で示された）コードを挿入します。

```
// Save the preferences.
PortletPreferences prefs = request.getPreferences();
String param = request.getParameter(PORTLETTITLE_KEY);
prefs.setValues(PORTLETTITLE_KEY, buildValueArray(param));
String contentParam = request.getParameter("portletContent");
prefs.setValues("portletContent", buildValueArray(contentParam));
prefs.store();
```

14. **「すべて保存」**アイコンをクリックして、view.jsp、edit.jsp、edit_defaults.jsp および MyJavaPortlet.java への変更を保存します。

    いくつか変更を行ったので、Preconfigured OC4J にポートレットを再デプロイする必要があります。

15. アプリケーション・ナビゲータで、「Portlets」、「リソース」の下の**「TutorialPortlets.deploy」**を右クリックします。

16. **「MyLocalOC4J に配布」**をクリックします。

17. **「OK」**をクリックして、「アプリケーションの構成」ダイアログを閉じます。

18. TutorialPortlets.deploy はすでにデプロイされているので、**「はい」**をクリックして、元のバージョンをアンデプロイすることを確認します。

    「デプロイが終了」というメッセージが表示されるのを待ちます。JDeveloper によって、ポートレットのデプロイ前に、コードが自動的に保存およびコンパイルされることがわかります。

19. 再度ページを実行する前に、埋込み OC4J サーバーを停止します。メイン・メニューから**「実行」**、**「終了」**、**「埋込み OC4J サーバー」**の順に選択します。

    あるいは、埋込み OC4J サーバーのログ・ウィンドウで、赤い四角形の「終了」アイコンをクリックします。ページをテストする前には、埋込み OC4J を停止することをお薦めします。ページを実行すると、JDeveloper によって埋込み OC4J サーバーが自動的に再起動されます。

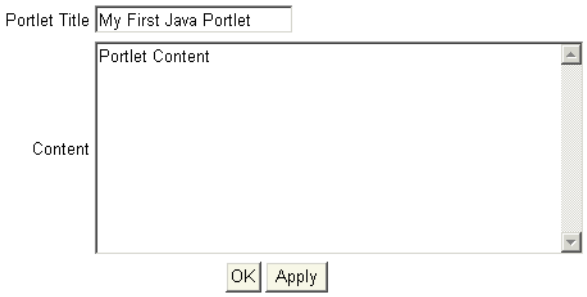
20. ビジュアル・エディタに MyPage.jspx が表示されることを確認してから、**「実行」**→**「MyPage.jspx」**をクリックします。

    新しいブラウザ・ウィンドウが開き、MyJavaPortlet が表示されます。

21. ドロップダウン・メニューから、**「カスタマイズ」**を選択します。

    「コンテンツ」フィールドを追加したので、「カスタマイズ」アクションでは図 3-43 のようなテキスト・ボックスが表示されます。

図 3-43 新しい「カスタマイズ」オプション

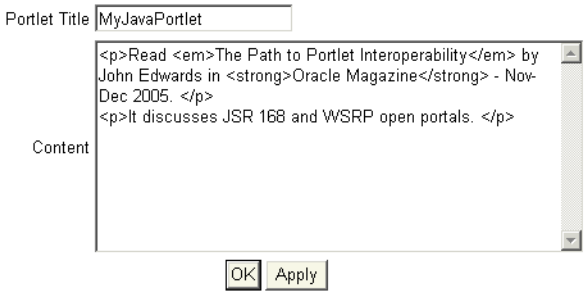


22. 「ポートレット・タイトル」を「MyJavaPortlet」に戻します。

23. 「コンテンツ」フィールドに次の HTML テキストをコピーして貼り付けます（図 3-44 を参照）。

```
<p>Read <em>The Path to Portlet Interoperability</em> by John Edwards in
<strong>Oracle Magazine</strong> - Nov-Dec 2005. </p>
<p>It discusses JSR 168 and WSRP open portals. </p>
```

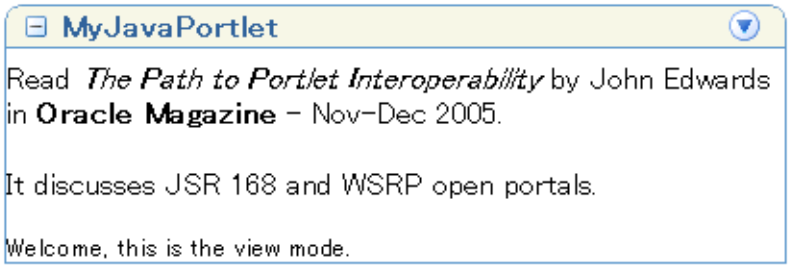
図 3-44「ポートレット・タイトル」および「コンテンツ」のカスタマイズ



24. 「**OK**」をクリックします。

ブラウザは図 3-45 のように表示されます。

図 3-45 MyJavaPortlet の新しい HTML テキスト



デフォルト・テキスト（Welcome, this is the view mode.）を削除する場合は、「**view.jsp**」をダブルクリックし、コードの最後の行を削除し、Preconfigured OC4J にポートレットを再デプロイしてから、ページを再実行するだけです。

次のレッスンでは、イメージ・フォームのコンテンツをページに追加します。

4

ページのカスタマイズ

このレッスンでは、前のレッスンで作成したページを拡張します。具体的には、カスタマイズ可能なコンポーネントをページに追加し、コンポーネントのルック・アンド・フィールに影響を与える様々な方法を試してみます。その過程で、ページをテストして、ページの開発が予測どおりに進んでいることを確認します。

図 4-1 に、このレッスンを終えた時点での、ページ上のコンポーネントのレイアウトを示します。

図 4-1 レッスン 4 を終えた時点での MyPage.jspx

概要

次の手順を実行することによって、ページをカスタマイズします。

- 手順 1: ユーザーにページのカスタマイズを許可
- 手順 2: ページの実行およびカスタマイズ
- 手順 3: 追加のカスタマイズ
- 手順 4: 新しいカスタマイズのテスト
- 手順 5: ルック・アンド・フィールの変更

まずは、Oracle WebCenter Framework で提供されるいくつかのメリットについて説明します。

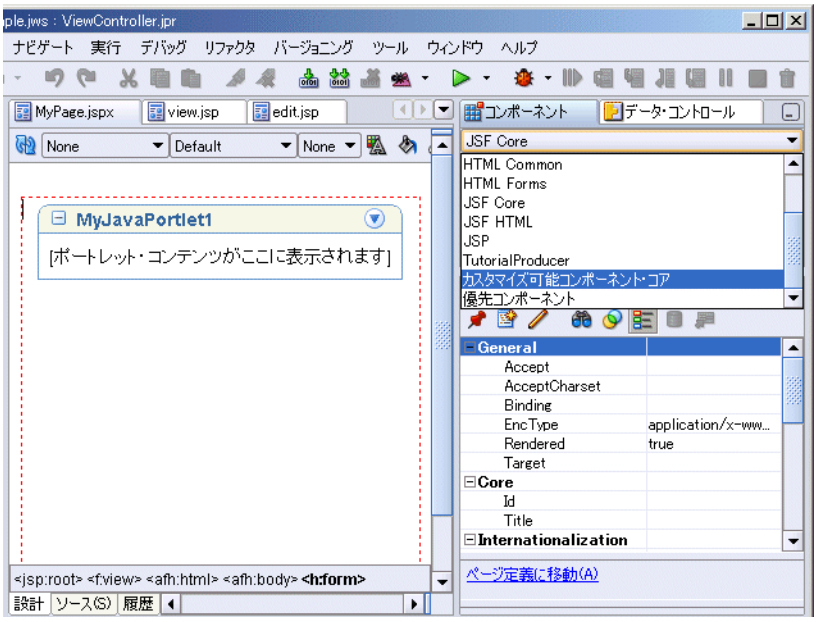
手順 1: ユーザーにページのカスタマイズを許可

Oracle WebCenter Framework の主要な機能の 1 つは、サイト管理者が他の人のページをカスタマイズできる「実行時での設計時」という概念です。カスタマイズは、カスタマイズ可能なコンポーネント（具体的には `PanelCustomizable` および `ShowDetailFrame`）を使用して行います。（ページのカスタマイズは、前の章でポートレット・アクションを使用して行ったポートレットのカスタマイズとは異なります。）

この 2 つのコンポーネントを使用すると、コンテンツの特定の部分を非表示にしたり、ページの別の位置に移動したりするためのオプションをユーザーに与えることができます。これから、その方法を示します。

1. ビジュアル・エディタに MyPage.jspx が表示されることを確認します。
2. コンポーネント・パレットを開きます。
3. ドロップダウン・リストを使用して、**「カスタマイズ可能コンポーネント・コア」**を選択します（図 4-2）。

図 4-2 カスタマイズ可能コンポーネント・コア



このグループには次の 2 つの項目があります。

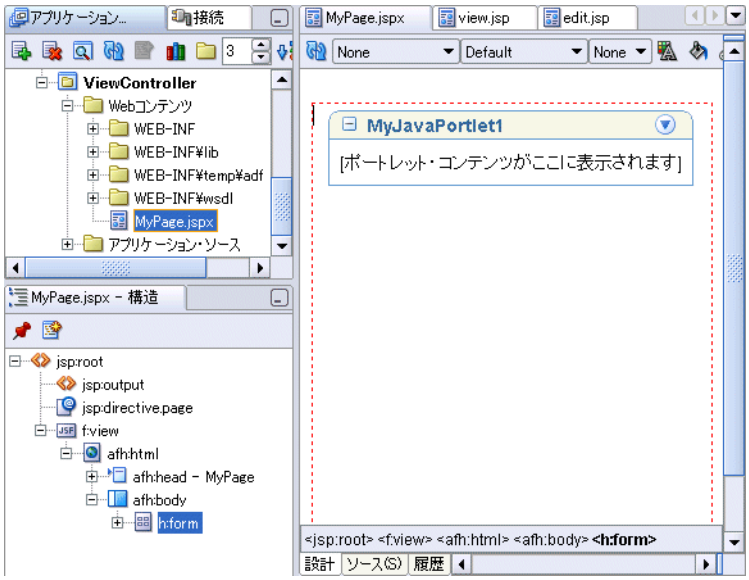
- **PanelCustomizable:** これは、Oracle ADF コンポーネントを含める水平レイアウトまたは垂直レイアウトを提供します。プロパティ・インスペクタでいくつかの属性を設定することにより、レイアウト内に配置するコンテンツを表示または非表示にするためのアクションを有効化できます。
- **ShowDetailFrame:** これを使用すると、特定のコンポーネントに対して許可されるアクションを表示できます。Oracle WebCenter Framework のこのリリースでは、ユーザーはコンテンツを移動、最小化または最大化できます。ShowDetailFrame は、クロム（ページ上のコンポーネントを囲む境界）として考えることができます。

これから、これらの概念が実際にどのように作用するかを示します。

4. アプリケーション・ナビゲータで、「**MyPage.jspx**」を選択します。

   ページの構造が、構造ウィンドウに表示されることがわかります（図 4-3）。

図 4-3 MyPage.jspx の構造



5. コンポーネント・パレットで、「**PanelCustomizable**」を選択して構造ウィンドウにドラッグします。これを「`h:form`」タグの上にドロップします。

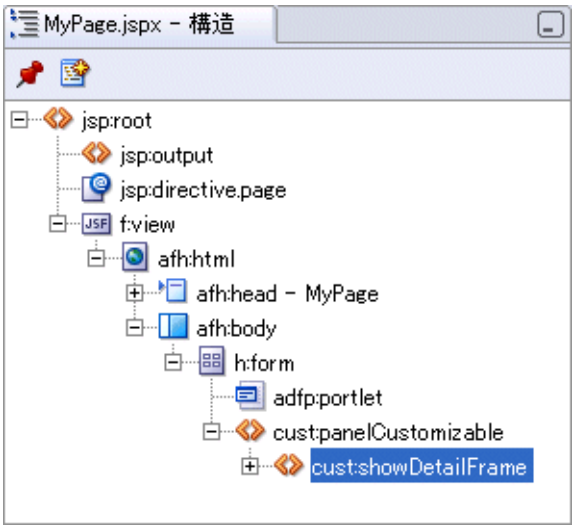
   ヒント： コンポーネントをフォーム内にドロップするときは、`h:form` を囲む枠があることを確認してください。

「構造」ビューで作業すると、コンポーネントをはるかに正確に配置できますが、コンポーネントを直接ビジュアル・エディタにドラッグ・アンド・ドロップすることもできます。

6. コンポーネント・パレットで、「**ShowDetailFrame**」を選択して構造ウィンドウにドラッグします。これを「`cust:panelCustomizable`」タグの上にドロップします。

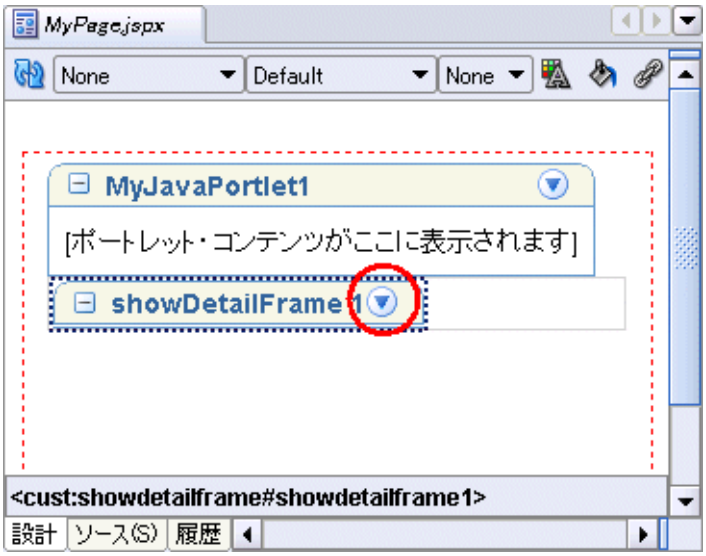
   「`cust:showDetailFrame`」は、「`cust:panelCustomizable`」の子になります（図 4-4 を参照）。

図 4-4 構造ウィンドウ内の階層



ビジュアル・エディタ（図 4-5）で、グレーのボックス内に「`showDetailFrame1`」が表示されます。グレーのボックスは `cust:panelCustomizable` を表すため、この中に `showDetailFrame` を適切にドロップしたことがわかります。丸で囲まれた領域は、ユーザーが `showDetailFrame` のコンテンツのカスタマイズに使用するコントロールを表します。

図 4-5 MyPage.jspx に追加された ShowDetailFrame コンポーネント



今度は、簡単なイメージのフォームのコンテンツを追加します。

7. コンポーネント・パレットのドロップダウン・リストで、「**ADF Faces Core**」を選択します。

   すると、一般的に使用される ADF コンポーネントのリストが表示されます。

8. アルファベット順リストをスクロールダウンして、「**ObjectImage**」コンポーネントを見つけます。

9. 「**ObjectImage**」を選択して構造ウィンドウにドラッグします。これを「`cust:showDetailFrame`」の上にドロップします。

   ObjectImage の挿入ウィンドウが開きます。

10. 「**...**」ボタンを使用して、`webcentertutorialcontent.zip` を解凍したディレクトリを見つけます（「サンプル・チュートリアル・ファイルのダウンロード」を参照）。たとえば、`C:¥TutorialContent` などです。

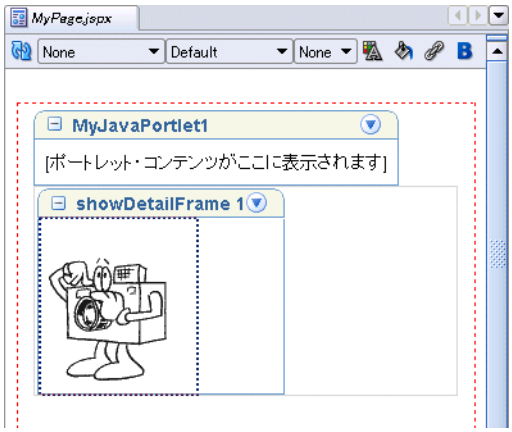
11. イメージ「`camera.gif`」を選択し、「**OK**」をクリックします。

    ドキュメント・ルートの下にイメージを配置するかどうか尋ねるダイアログが開きます。「はい」を選択すると、イメージのコピーが、アプリケーション・デプロイメント・アーカイブの作成時に自動的に含められる場所に保存されます。

12. 「**はい**」をクリックします。

13. 「**保存**」をクリックしてから「**OK**」をクリックします。

図 4-6 MyPage.jspx へのイメージの追加



イメージ `camera.gif` が、`showDetailFrame` コンポーネント内に表示されます（図 4-6 を参照）。

> **ヒント：** イメージは、ファイル・システム（たとえば、C:¥TutorialContent¥camera.gif など）から `ShowDetailFrame` コンポーネントに直接ドラッグ・アンド・ドロップできます。ObjectImage が自動的に作成され、イメージがプロジェクトにコピーされます。

今度は、別のイメージを追加して、ユーザーがページ上でイメージを移動する方法を学びます。

14. コンポーネント・パレットのドロップダウン・リストで、「**カスタマイズ可能コンポーネント・コア**」を選択し、「**ShowDetailFrame**」を選択します。

15. 「**ShowDetailFrame**」を構造ウィンドウにドラッグし、「`cust:panelCustomizable`」の上にドロップします。

16. コンポーネント・パレットのドロップダウン・リストで、「**ADF Faces Core**」を選択し、「**ObjectImage**」を選択します。

17. 「**ObjectImage**」を構造ウィンドウにドラッグします。これを 2 つ目の「`cust:showDetailFrame`」の上にドロップします。
18. 「...」ボタンを使用して `hula.gif` を見つけ（「サンプル・チュートリアル・ファイルのダウンロード」を参照）、「**OK**」をクリックします。

    ドキュメント・ルートの下にイメージを配置するかどうか尋ねるダイアログが開きます。
19. 「**はい**」を選択します。
20. 「**保存**」をクリックしてから「**OK**」をクリックします。

    ビジュアル・エディタでの「設計」タブは、図 4-7 のように表示されます。

図 4-7 MyPage.jspx 上の 2 つのイメージおよび 1 つのポートレット

各イメージには `showDetailFrame` ヘッダーにそれぞれアクションが関連付けられているため、互いに無関係に動作できます。

今度は、ページを実行し、どのようなカスタマイズが可能かを確認します。

手順 2: ページの実行およびカスタマイズ

この手順では、ページを実行し、ユーザーが行えるカスタマイズをテストします。

1. 「**MyPage.jspx**」を右クリックし、「**実行**」をクリックします。

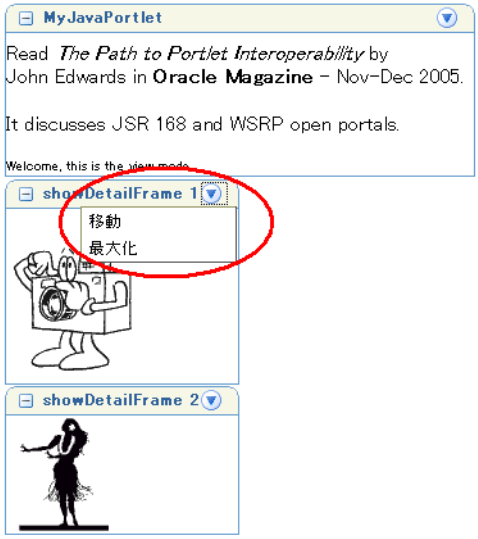
   コンパイラ・エラーがなければ、ページが別のブラウザ・ウィンドウ内に開きます（図 4-8 を参照）。

図 4-8 2 つのイメージを持つ MyPage.jspx

2. 最初のイメージの**アクション**・アイコンをクリックします。

   2 つのアクション（「移動」および「最大化」）が表示されます（図 4-9 を参照）。

図 4-9 アクション・メニュー

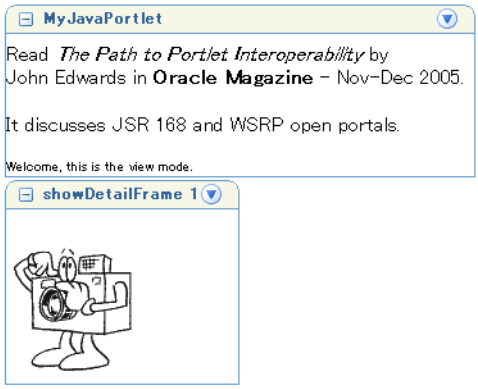


これらのアクションは、`showDetailFrame` によって自動的に提供されます。`showDetailFrame` 内にドロップしたコンポーネントはいずれも、その親である `panelCustomizable` のコンテキストで移動または最大化できます。

3. 最初のイメージ内で**「最大化」**をクリックし、その `showDetailFrame` が拡大されて `panelCustomizable` いっぱいに表示されることを確認します（図 4-10 を参照）。2 つ目のイメージは非表示になります。

   MyJavaPortlet はこのアクションに影響されないことに注意してください。これは、ポートレットがこの `panelCustomizable` の一部ではないためです。

図 4-10 最大化された 1 つのイメージ



4. **アクション・**メニューをクリックし、**「リストア」**をクリックして、イメージを元のサイズに戻します。

5. いずれかのイメージに対して**「移動」**、**「下へ」**または**「移動」**、**「上へ」**をクリックして、ページ上のイメージ位置が入れ替わることを確認します。ここでも、MyJavaPortlet はこの panelCustomizable の制御下にはないため、イメージの位置が入れ替わっても MyJavaPortlet の位置は変わりません。

   図 4-11 に、最初の図形（カメラ）に対して**「移動」**、**「下へ」**を使用した後のブラウザを示します。フラの図形は、カメラの図形の上に配置されています。

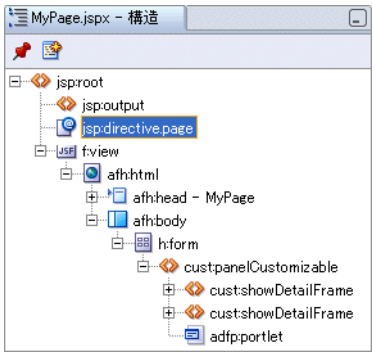
図 4-11 イメージの移動

次の手順では、MyJavaPortlet を「`cust:panelCustomizable`」に移動し、プロパティ・インスペクタを使用してページ上の様々なコンポーネントの動作を試してみます。

手順 3: 追加のカスタマイズ

1. ブラウザを閉じ、JDeveloper に戻ります。
2. 構造ウィンドウで、(MyJavaPortlet を表す)「**adfp:portlet**」をドラッグし、「`cust:panelCustomizable`」にドロップします(図 4-12)。

図 4-12 ポートレットを PanelCustomizable にドロップ



ポートレットは、自動的にオブジェクト・リストの一番下に移動します。また、「設計」ビューでも同様になります。ポートレットは、2 つ目のイメージの下に表示されます。

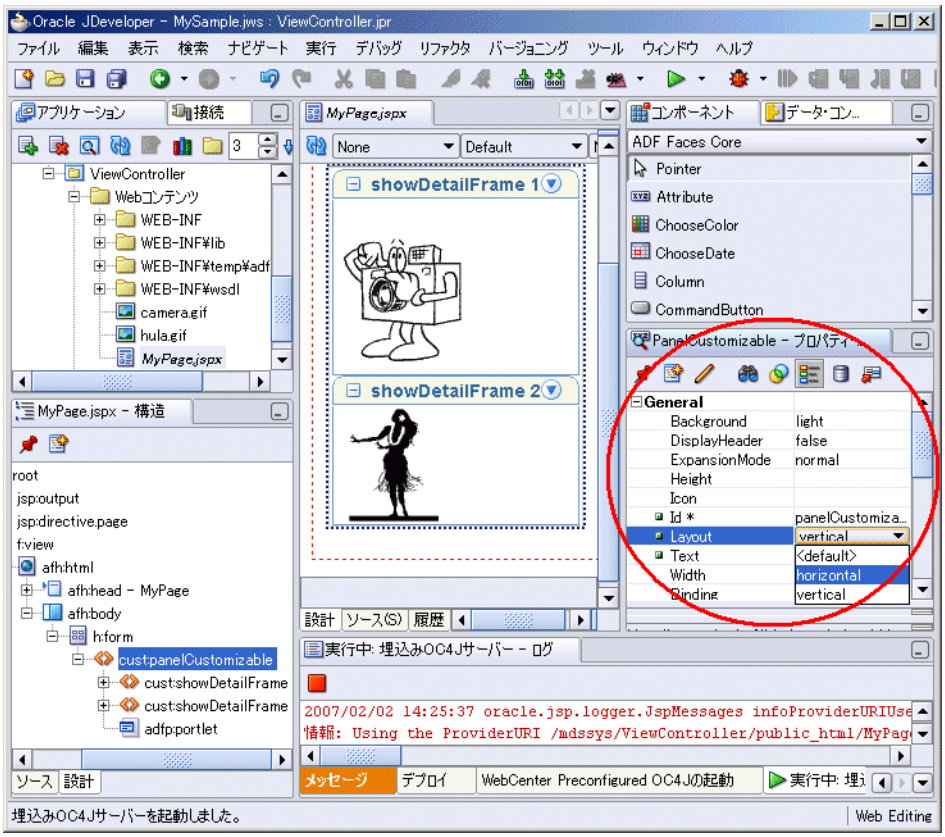
最初に `showDetailFrame` を挿入せずにポートレットを直接「`cust:panelCustomizable`」にドロップしたことに注意してください。これは、ポートレットには、ポートレット・クロムによって提供されるカスタマイズ機能(ページ上でポートレットの最大化 / 最小化や位置変更を行う機能)が自動的に備わっているためです。

今度は、ポートレットおよびイメージに別の位置決めを選択します。これらを垂直に表示せずに、水平に表示します。

3. 構造ウィンドウで、「`cust:panelCustomizable`」を選択します。

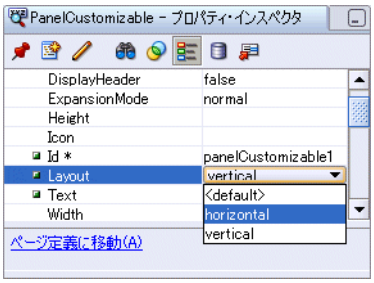
4. プロパティ・インスペクタの「General」で、**Layout** の隣のフィールドをクリックします。プロパティ・インスペクタは、JDeveloper 画面の右部分です（図 4-13）。

図 4-13 JDeveloper のプロパティ・インスペクタ



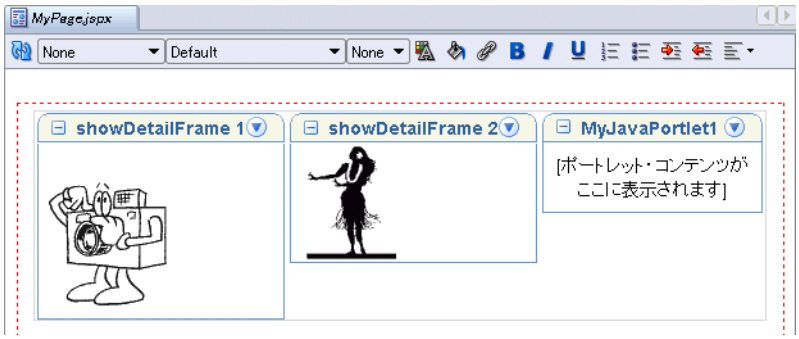
5. **horizontal** を選択します（図 4-14 を参照）。

図 4-14 水平レイアウトの選択



ビジュアル・エディタで、ポートレットとイメージが上下にではなく横に並んで表示されることがわかります（図 4-15 を参照）。

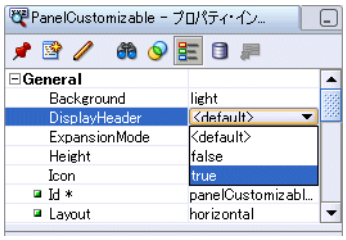
図 4-15 水平レイアウト表示



今度は、コンテンツを表示または非表示にする機能を有効にします。

6. プロパティ・インスペクタの「General」で、**DisplayHeader** を見つけて **true** に変更します（図 4-16 を参照）。

図 4-16 DisplayHeader を true に変更



これによって、メニューの一部としてアクション・アイコンが表示されるためのヘッダー領域が、レイアウトの一番上に表示されます。（独自のアプリケーションでヘッダーを表示しない場合は、DisplayHeader を false のままにしておくことができます。）次の手順を実行する間は、`panelCustomizable` 領域の上にカーソルを合せると、コンポーネントのメニュー・アクションが表示されます。

7. プロパティ・インスペクタの「アクション」で、IsSeededInteractionAvailable を見つけて **true** に変更します。

   IsSeededInteractionAvailable 属性を設定すると、コンポーネントによってサポートされるメニュー・アクションを表示することが可能になります。この場合、ユーザーは、レイアウト内に表示されるコンテンツを表示または非表示にすることができます。

   今度は、プロパティ・インスペクタを使用して MyJavaPortlet の名前を変更する方法を学びます。

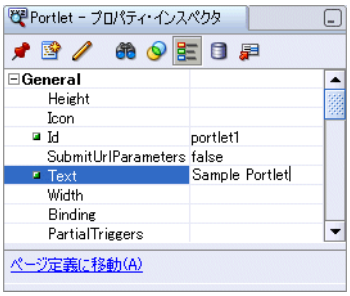
   プロパティ・インスペクタを介して変更を行った後、実行時にポートレット名を再度カスタマイズまたはパーソナライズすることはできません。この動作はこのチュートリアルの目的上は問題ありませんが、将来的に独自のアプリケーションを開発する際には留意しておく必要があります。

8. プロパティ・インスペクタにポートレットのプロパティが反映されるように、構造ウィンドウで「**adfp:portlet**」を選択します。

9. プロパティ・インスペクタの「General」で、Text フィールドを見つけ、`Sample Portlet` を入力します（図 4-17 を参照）。

   これによって、ポートレット・ヘッダーに表示される名前が変更されます。

図 4-17 ポートレットの名前変更



10. JDeveloper のツールバーで、**「すべて保存」**をクリックします。

これから、再度ページを実行して、これらの変更がページにどのような影響を及ぼすかを確認します。

手順 4: 新しいカスタマイズのテスト

カスタマイズをテストするには、ページを実行し、ページが予測どおりに表示されることを確認します。

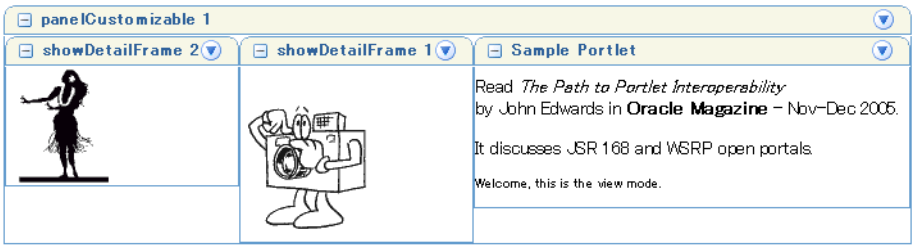
1. 再度ページを実行する前に、埋込み OC4J サーバーを停止します。メイン・メニューから**「実行」**、**「終了」**、**「埋込み OC4J サーバー」**の順に選択します。

   あるいは、埋込み OC4J サーバーのログ・ウィンドウで、赤い四角形の「終了」アイコンをクリックします。

2. ビジュアル・エディタで**「MyPage.jspx」**をクリックし、ツールバーの緑色の矢印をクリックしてページを実行します。

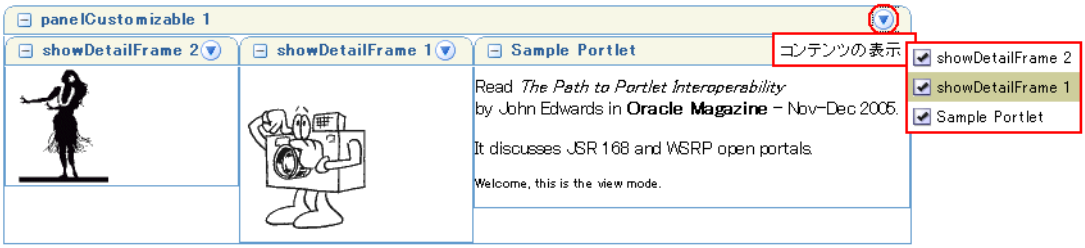
   コンパイラ・エラーがなければ、ページが新しいブラウザ・ウィンドウ内に開きます（図 4-18）。

図 4-18 2 つの showDetailFrame および 1 つのポートレットを持つ panelCustomizable



3. IsSeededInteractionAvailable をアクティブにした結果、コンテナ・オブジェクトのアクション・アイコンには**「コンテンツの表示」**アクションが表示されます。このアクションを使用すると、イメージとポートレットを非表示にしたり表示することができます（図 4-19）。ここで、これらの操作を試してみます。

図 4-19「コンテンツの表示」アクション



手順 3 で行った他のページ・カスタマイズもすべて反映されています。

- ポートレットのタイトルは、Sample Portlet に変更されています。ポートレット・タイトルは Text プロパティの影響を受けているため、実行時にタイトルをカスタマイズまたはパーソナライズすることはできません。アクション・アイコンをクリックして「カスタマイズ」をクリックすると、これがわかります。
- コンポーネントは、垂直ではなく水平に表示されています。イメージの 1 つに対してアクション・アイコンをクリックすると、「移動」アクションに「上へ」と「下へ」ではなく「左」と「右」が表示されることがわかります。
- DisplayHeader を true に変更したため、コンテナ・オブジェクトの panelCustomizable にヘッダー領域が表示されています。DisplayHeader 設定を変更する前、手順 3 ではこのようなヘッダーはレイアウトになかったことを思い出してください。

ページを実行した人には、これらのページ・カスタマイズが表示されます。

4. 先に進む前に、プロパティ・インスペクタで行ったポートレット・タイトルの永続的カスタマイズを削除します。第 8 章「セキュリティの設定」では、実行時に名前を変更できるように、ポートレットの title プロパティを使用してページをパーソナライズする例を示します。
   a. ブラウザを閉じ、JDeveloper に戻ります。
   b. MyPage.jpsx の構造ウィンドウで、「`adfp:portlet`」を選択します。
   c. プロパティ・インスペクタで **Text** フィールドをクリックし、プロパティ・インスペクタのツールバーで**「デフォルトにリセット」**アイコンをクリックします。これによって、テキスト Sample Portlet が削除されます。

      ビジュアル・エディタで、ポートレット・ヘッダーの名前が MyJavaPortlet1 に戻ります。

将来、1 つのページ上に複数の `panelCustomizable` コンポーネントを使用し、それぞれで、1 つの `showDetailFrame` コンポーネント内に複数のポートレットまたはオブジェクトを含めることができます。各レイアウトにそれぞれの「移動」、「最大化」および「表示 / 非表示」機能があるため、ページに複数のレイアウトを配置すると、コンポーネントおよびポートレットのブロックを個別のエンティティとして扱うことができます。詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

手順 5: ルック・アンド・フィールの変更

Oracle WebCenter Framework では、アプリケーションのスタイル（つまり、ルック・アンド・フィール）を制御する方法が 2 つあります。

- Oracle ADF Faces スキンを使用して、グローバル・スタイルをアプリケーション全体に適用します。このオプションを選択した場合、独自のカスタム・スキンでスタイル・セレクタを使用して、コンポーネントやコンポーネント領域の選択した局面を変更できます。
- プロパティ・インスペクタを使用して、設計時にスタイル関連のプロパティを変更します。

次の各手順では、この両方の方法を使用して、MyPage.jspx のルック・アンド・フィールを変更します。

- 手順 5a: デフォルトの ADF Faces スキンを使用した showDetailFrame 背景の変更
- 手順 5b: カスタム・スキンを使用した独自のスタイルの適用

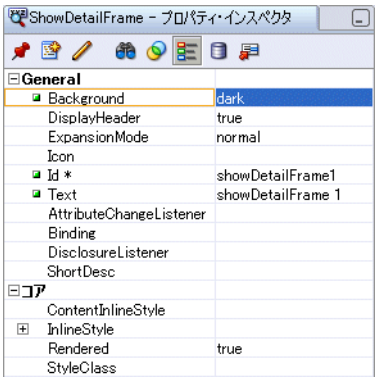
手順 5a: デフォルトの ADF Faces スキンを使用した showDetailFrame 背景の変更

この手順では、（Oracle という名前の）デフォルトの ADF Faces スキンを使用して、スタイル関連のプロパティを操作します。

1. 構造ウィンドウで、「`cust:showDetailFrame`」エントリの 1 つを選択します。このコンポーネントを使用して、スタイル・プロパティ値を指定する方法を示します。
2. プロパティ・インスペクタで、**Background** プロパティを開きます。

   デフォルトの ADF Faces スキンには、light、medium および dark の 3 つの設定があります。このコンポーネントには濃い背景色を選択します。
3. 値を **dark** に変更します（図 4-20 を参照）。

図 4-20 プロパティ・インスペクタ : Background

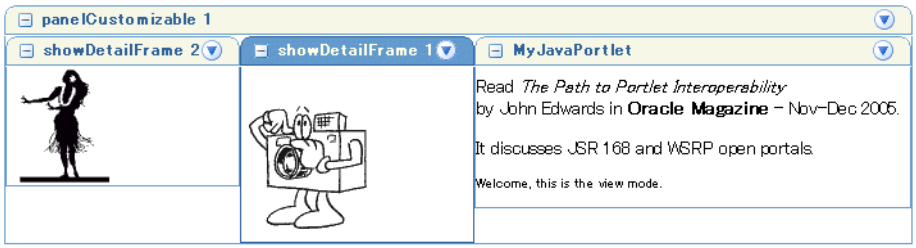


4. 「**ファイル**」、「**すべて保存**」の順にクリックして、作業を保存します。
5. 再度ページを実行する準備として、赤い四角形をクリックして、埋込み OC4J を停止します。

6. 「**MyPage.jspx**」を右クリックし、「**実行**」をクリックします。

   コンパイラ・エラーがなければ、ページが新しいブラウザ・ウィンドウ内に開きます（図 4-21）。`showDetailFrame` の背景の装飾が変更されたことに注意してください。

図 4-21 濃い背景色のイメージ



今度は、新しいスキンを作成し、デフォルトの ADF Faces スキンで定義されているスタイルをオーバーライドします。

手順 5b: カスタム・スキンを使用した独自のスタイルの適用

この手順では、新しいスキンを作成してアプリケーションに登録し、カスタム・スタイルを MyPage 上のコンポーネントに適用する方法を学びます。

1. まず、JDeveloper に戻り、新しいスタイルシート（.css）を作成します。
   a. アプリケーション・ナビゲータで、「**ViewController**」を右クリックし、「**新規**」を選択します。
   b. 新規ギャラリの「カテゴリ」で、「Web Tier」を開き、「**HTML**」を選択します。
   c. 「項目」で「**CSS ファイル**」を選択し、「**OK**」をクリックします。
   d. 「ファイル名」に、`mystyle.css` を入力します（図 4-22 を参照）。

図 4-22 Cascading Style Sheet の作成

   e. 「**OK**」をクリックします。

新しいスタイルシートが、アプリケーション・ナビゲータの「ViewController」、「Web コンテンツ」、「css」の下に表示されます。また、エディタにも表示されます。デフォルトのスタイル・セレクタ BODY、H1、H2 および H3 に注意してください。

以降の手順で、新しいスキンをアプリケーションに登録します。これには、adf-faces-skins.xml という名前のファイルを作成し、スキンの ID、場所、.css などを示すタグの短いリストをこのファイルに移入します。

2. adf-faces-skins.xml を作成します。
   a. アプリケーション・ナビゲータで、「ViewController」、「Web コンテンツ」の下の**「WEB-INF」**フォルダを右クリックし、**「新規」**を選択します。
   b. 新規ギャラリで、「フィルタ方法」スコープを**「すべてのテクノロジ」**に設定します。
   c. 「General」ノードで、**「XML」**を選択します。
   d. 右ペインで、**「XML 文書」**を選択し、**「OK」**をクリックします。
   e. 「ファイル名」フィールドに、ファイル名として adf-faces-skins.xml を入力します（図 4-23 を参照）。

図 4-23 adf-faces-skins.xml の作成

   f. このファイルを WEB-INF フォルダに格納するため、**「OK」**をクリックしてファイルを作成します。

      エディタに空の XML ファイルが表示されます。このファイルは、アプリケーション・ナビゲータで「WEB-INF」の下に表示されています。

3. タグを adf-faces-skins.xml に追加して、新しいスキンを指定します（また、新しいスタイルシート mystyle.css をポイントします）。
   a. 次のコードをコピーして XML エディタに貼り付けます。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://xmlns.oracle.com/adf/view/faces/skin">
 <skin>
  <id>mystyle</id>
  <family>mystyle</family>
  <render-kit-id>oracle.adf.desktop</render-kit-id>
  <style-sheet-name>css/mystyle.css</style-sheet-name>
 </skin>
</skins>
```

ファイルは、図 4-24 のように表示されます。

図 4-24 adf-faces-skins.xml の構成

b. **「ファイル」、「保存」**の順にクリックして、`adf-faces-skins.xml` を保存します。

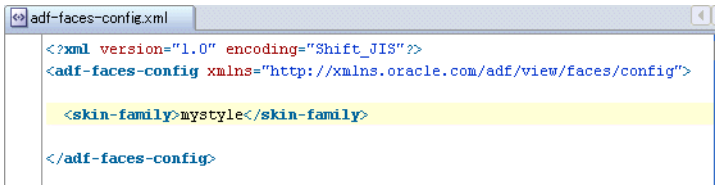
次の手順では、`adf-faces-config.xml` 内に <skin-family> タグを設定することにより、この新しいスキンを使用するようにアプリケーションを構成します。

4. `adf-faces-config.xml` 内の <skin-family> タグを編集します。

a. 「ViewController」、「Web コンテンツ」、「WEB-INF」の下にある「`adf-faces-config.xml`」を開きます。

b. `oracle`（デフォルトのスキン）を新しいスキンのファミリ名に置き換えます。`mystyle` を入力します（図 4-25 を参照）。

図 4-25 adf-faces-config.xml でのスキン・ファミリの編集



c. **「ファイル」、「保存」**の順にクリックします。

これで、新しいスキンがアプリケーションに登録されました。今度は、MyPage 上の showDetailFrame コンポーネントに適用するスタイルシートにいくつかのスタイル・セレクタを追加します。

5. 次のスタイル・セレクタを `mystyle.css` に追加します。

a. アプリケーション・ナビゲータの「ViewController」、「Web コンテンツ」、「css」の下の**「mystyle.css」**をダブルクリックします。

b. 次のコードをコピーしてファイルの一番下に貼り付けます。

```
af|showDetailFrame::header-light
{
 color:Purple;
}
af|showDetailFrame::header-medium
{
 color:Purple;
}
af|showDetailFrame::header-dark
{
 color:White;
}
```

このコードは、showDetailFrame コンポーネントのヘッダーおよびポートレットのヘッダーに表示されるテキストの色を指定するものです。ここでは、header-light および header-medium に purple のテキストを選択し、header-dark に white のテキストを選択しました。適宜、別のカラー・スキームを選択してください。

c. 次のコードをコピーしてファイルの一番下に貼り付けます。

```
af|showDetailFrame::content-light, af|showDetailFrame::content-medium,
af|showDetailFrame::content-dark
{
 color: Black;
 background-color: Silver;
 border-left:1px black solid;
 border-right:1px black solid;
 border-bottom:1px black solid;
}
```

このコードは、showDetailFrame コンポーネントおよびポートレット本体のコンテンツについて、背景を銀色、テキストを黒色、境界線を黒色に変更するものです。適宜、別のカラー・スキームを選択してください。

showDetailFrame コンポーネントおよび panelCustomizable コンポーネントのスタイル・セレクタの完全なリストは、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

6. 今度は、ページの背景色を変更します。スタイルシートの一番上にある BODY タグにスクロールし、背景色を White から Purple に変更します。BODY タグは次のようになります。

```
BODY
{
   background-color: Purple;
   color: black;
   font-family: Arial, Helvetica, sans-serif;
}
```

7. 「ファイル」、「すべて保存」の順にクリックして、作業を保存します。

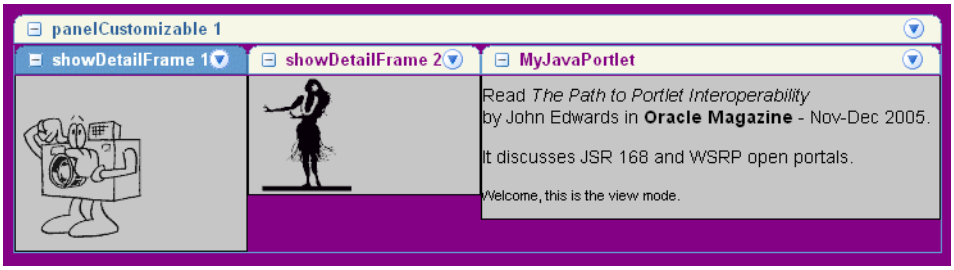
8. 再度ページを実行する準備として、赤い四角形をクリックして、埋込み OC4J を停止します。

9. 「MyPage.jspx」を右クリックし、「実行」をクリックします。

コンパイラ・エラーがなければ、ページが新しいブラウザ・ウィンドウ内に開きます。

ここで、新しいルック・アンド・フィールを調べてみます。ここに示したスタイルを使用した場合、ページは図 4-26 のように表示されます。showDetailFrames で、紫色と白色のヘッダー・テキスト、黒色のコンテンツ・テキスト、および銀色の背景が表示されていることに注意してください。また、ページの背景も紫色になります。

図 4-26 mystyle.css を使用する MyPage.jspx



このレッスンでは、アプリケーションのルック・アンド・フィールを操作する方法を試してみました。詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

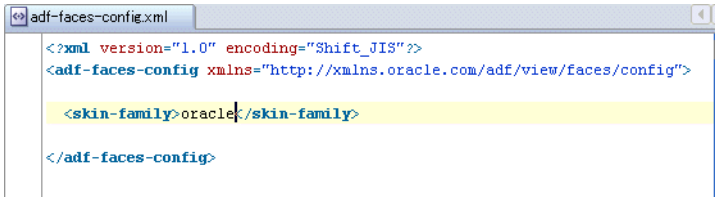
先に進む前に、デフォルトの Oracle スキンに戻します。

10. カスタム・スキンを削除するには、次のようにします。

    a. 「ViewController」、「Web コンテンツ」、「WEB-INF」の下にある「`adf-faces-config.xml`」を開きます。

    b. <skin-family> タグを `oracle` に戻します（図 4-27 を参照）。

図 4-27 adf-faces-config.xml でのスキン・ファミリの編集



    c. **「ファイル」、「すべて保存」**の順にクリックします。

次のレッスンでは、ユーザーへの掲示板のような機能を提供するリッチ・テキスト・ポートレットを追加する方法を学びます。

5

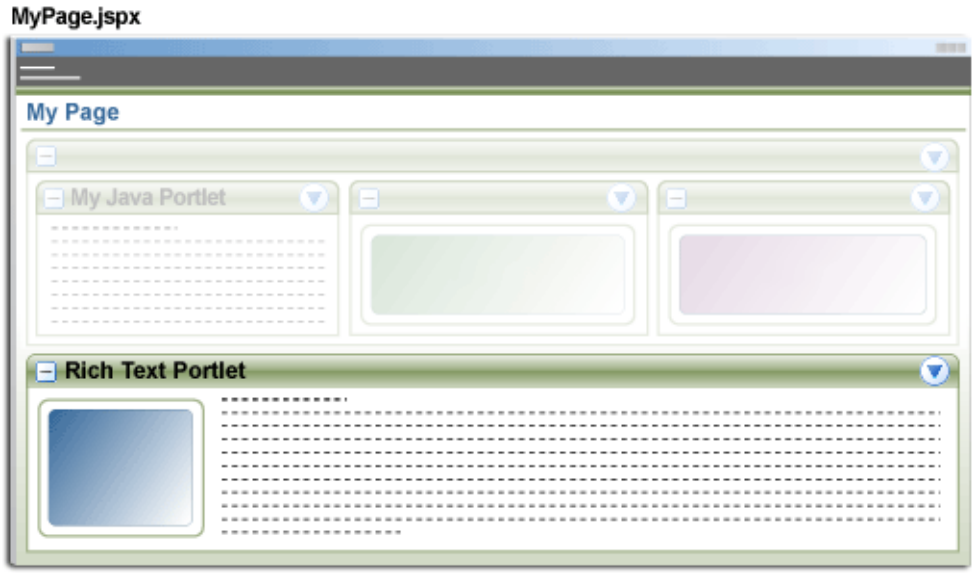
リッチ・テキスト・ポートレットの追加

リッチ・テキスト・ポートレットは、企業の告知やニュースのアイテムを実行時に掲載するための便利なツールです。このポートレットをページに追加すると、認可されたユーザーは、アクション・アイコンを使用して、表示テキストの挿入、更新および書式設定に必要なすべてのリッチテキスト編集ツールが備わったツールバーを起動できます。リッチ・テキスト・ポートレットを使用すると、ページに設定されているセキュリティ権限に応じて、幅広い読者または範囲の狭い定義済グループに対して情報を配信できます。

リッチ・テキスト・ポートレットは、WebCenter Preconfigured OC4J を介して使用可能です。

図 5-1 に、このレッスンを終えた時点でのページの外観を示します。

図 5-1 レッスン 5 を終えた時点での MyPage.jspx



概要

このレッスンでは、次の手順を実行することにより、リッチ・テキスト・ポートレットでの作業方法を示します。

- 手順 1: リッチ・テキスト・プロデューサの登録
- 手順 2: ページへのリッチ・テキスト・ポートレットの追加
- 手順 3: 実行時のリッチ・テキスト・ポートレットのカスタマイズ

前提条件

リッチ・テキスト・ポートレットは、（第 3 章の「手順 2: 接続の設定」でインストールおよび初期化した）WebCenter Preconfigured OC4 を介して使用可能です。この OC4J インスタンスがもう実行されていない場合は、このレッスンを開始する前に、JDeveloper ツールバーの一番右にある**「WebCenter Preconfigured OC4J の起動」**をクリックします。

手順 1: リッチ・テキスト・プロデューサの登録

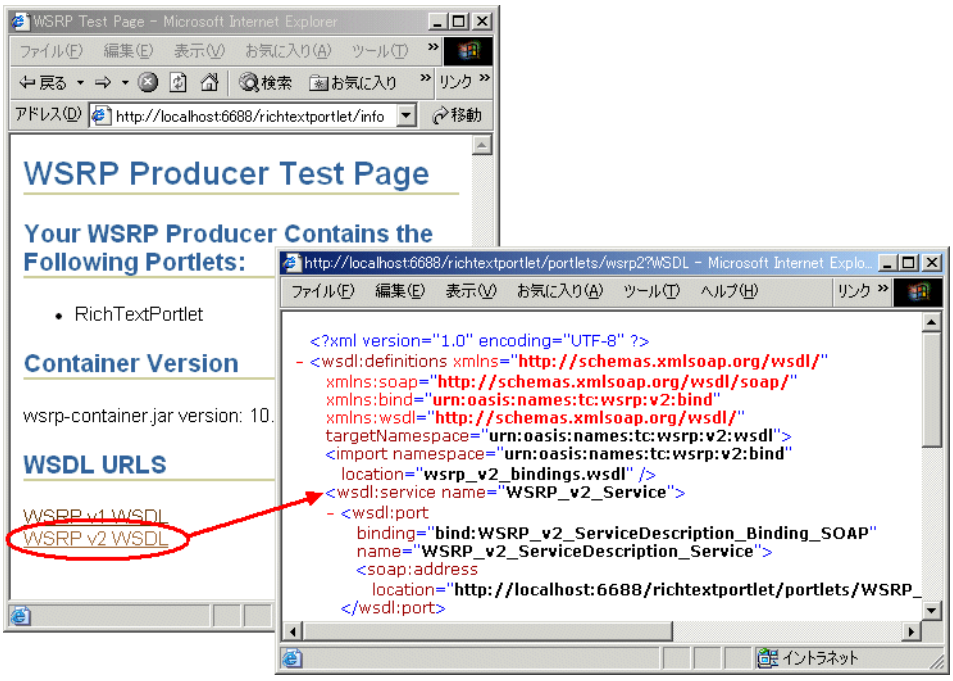
この手順では、リッチ・テキスト・プロデューサをアプリケーションに登録します。しかしまずは、リッチ・テキスト・プロデューサが WebCenter Preconfigured OC4J を介して使用可能であることを確認します。

1. ブラウザで、リッチ・テキスト・プロデューサのテスト・ページ URL を入力します。

   ```
   http://localhost:6688/richtextportlet/info
   ```

   プロデューサが稼働中の場合は、図 5-2 に示すような WSRP プロデューサ・テスト・ページが表示されます。

図 5-2 リッチ・テキスト・プロデューサのテスト・ページ



テスト・ページには、2 つのリッチ・テキスト・プロデューサ登録 URL（WSDL ドキュメント）が表示されます。この WebCenter アプリケーションでは、**WSRP v2 WSDL** の URL を使用します。

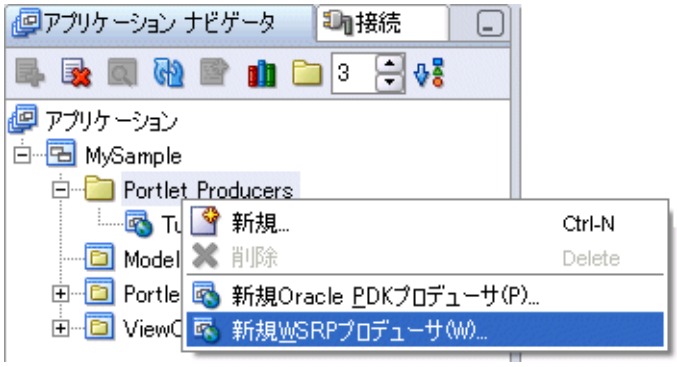
```
http://localhost:6688/richtextportlet/portlets/wsrp2?WSDL
```

ヒント： WebCenter Preconfigured OC4J を介して使用可能なあらゆるポートレット・プロデューサのテスト・ページは、`http://localhost:6688/` からアクセスできます。

次に、リッチ・テキスト・プロデューサを登録します。

2. アプリケーション・ナビゲータで、「**Portlet Producers**」を右クリックし、「**新規 WSRP プロデューサ**」をクリックします（図 5-3 を参照）。

図 5-3 アプリケーションへのリッチ・テキスト・ポートレットの登録



すでに 1 つポートレット・プロデューサを登録してあるので、「新規ギャラリ」にナビゲートしなくても、この右クリック・ショートカットを使用できます。

3. 「**次へ**」をクリックして、ウィザードの「ようこそ」画面を終了します。
4. 「名前」フィールドに `RichTextProducer` を入力します。
5. 「**次へ**」をクリックします。
6. 「URL エンドポイント」フィールドに、次のように入力します。

   ```
   http://localhost:6688/richtextportlet/portlets/wsrp2?WSDL
   ```

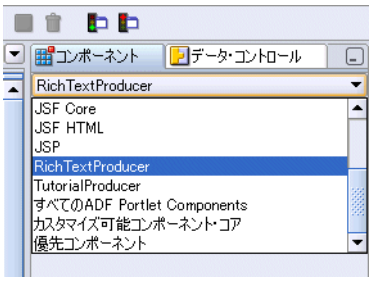
   このチュートリアルでは、URL 内に `localhost` を使用すると、Preconfigured OC4J がインストールされているローカル・コンピュータを参照できることを前提としています。（そうでない場合は、`localhost` をご使用のコンピュータの IP アドレスに置き換えてください。）
7. 「**次へ**」をクリックします。
8. 「**終了**」をクリックして、ウィザードを終了します。
9. 「**OK**」をクリックして、メッセージ・ボックスを閉じます。

   これで、リッチ・テキスト・ポートレットをページにドロップできるようになります。

手順 2: ページへのリッチ・テキスト・ポートレットの追加

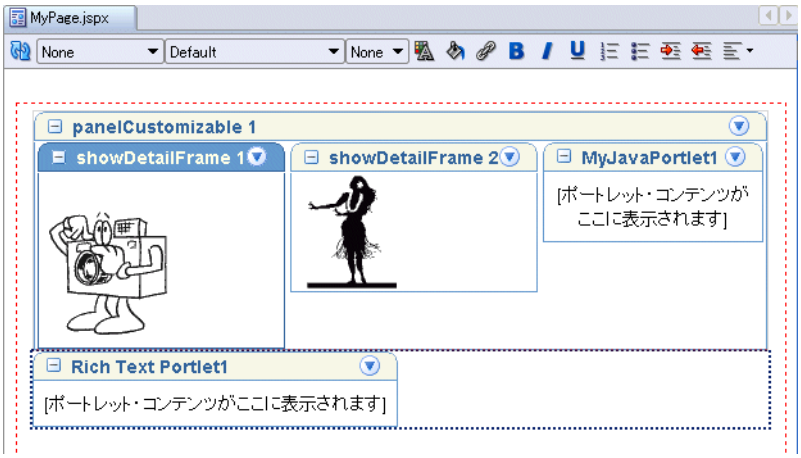
1. ビジュアル・エディタで MyPage.jspx を表示します。
2. コンポーネント・パレットから、「**RichTextProducer**」を選択します（図 5-4）。

図 5-4 リッチ・テキスト・ポートレット・プロデューサ



3. 「**リッチ・テキスト・ポートレット**」を選択し、構造ウィンドウ内の「`h:form`」の上にドロップします（図 5-5 を参照）。

図 5-5 ページへのリッチ・テキスト・ポートレットの追加



4. ビジュアル・エディタで「**Rich Text Portlet1**」を選択します。すると、プロパティ・インスペクタにそのプロパティが表示されます。
5. プロパティ・インスペクタの「表示オプション」で、AllModesSharedScreen を **true** に設定します。

   これによって、ページ全体をリフレッシュするのではなく、パーソナライズまたはカスタマイズを介して行ったすべての変更をインラインで表示するようにページを設定します。
6. 「**ファイル**」、「**すべて保存**」の順に選択して、作業を保存します。

7. **「実行」**、**「MyPage.jspx の実行」**の順に選択するか、JDeveloper ツールバーで緑色の矢印をクリックします。

   新しいブラウザが開き、ページが表示されます（図 5-6）。

図 5-6 デフォルトのリッチ・テキスト・ポートレット

ページは、ユーザーにこのように表示されます。次の手順では、リッチ・テキスト・ポートレットのカスタマイズが簡単であることを示します。WebCenter アプリケーションを計画するうえで役立つ点がいくつか示されています。

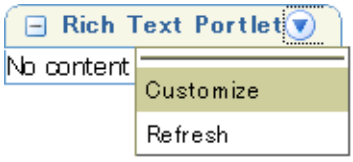
手順 3: 実行時のリッチ・テキスト・ポートレットのカスタマイズ

リッチ・テキスト・ポートレットをページに配置した後、ユーザーはアクション・アイコンを使用して、すべての人に表示されるテキストを入力できます。

1. リッチ・テキスト・ポートレットのヘッダーで、**アクション**・アイコンをクリックします。

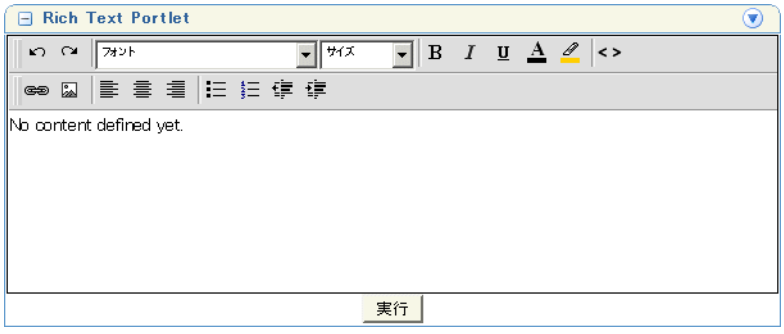
   リッチ・テキスト・ポートレットは、図 5-7 のように表示されます。

図 5-7 リッチ・テキスト・ポートレットのアクション・メニュー



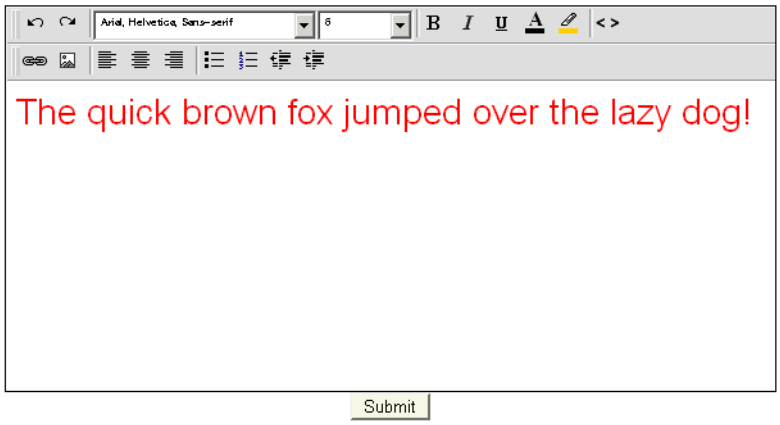
2. **「カスタマイズ」**をクリックして、リッチ・テキスト・コントロールを表示します（図 5-8）。

図 5-8 リッチ・テキスト・ポートレットのテキスト・コントロール



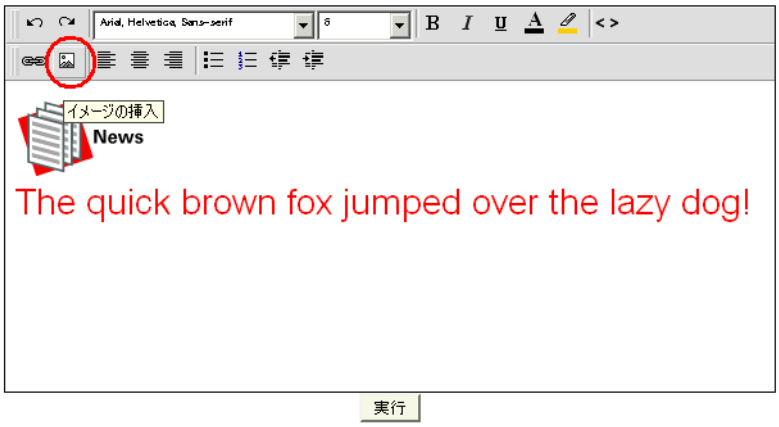
3. 独自のテキストを入力し、フォントのサイズ、タイプおよび色を変更し、背景色を選択します。例は、図 5-9 を参照してください。

図 5-9 Arial フォント、サイズ 6、赤色のテキスト



4. 他のコントロールも試してみます。たとえば、ポートレット内にイメージを表示します。
   a. 「**イメージの挿入**」アイコンをクリックします。
   b. イメージの URL を入力し、「**OK**」をクリックします（図 5-10）。

図 5-10 リッチ・テキスト・エディタを使用したイメージの挿入



5. 完了したら、「**発行**」をクリックします。

   ページがリフレッシュされて、新しいテキストおよびイメージが表示されます。

リッチ・テキスト・ポートレットおよびその他の事前パッケージ済ポートレットの詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

今度は、2 つのポートレット間の通信を有効にする方法、つまり、一方のポートレットからパラメータを渡して、もう一方のポートレットに表示されるコンテンツを制御する方法を学びます。

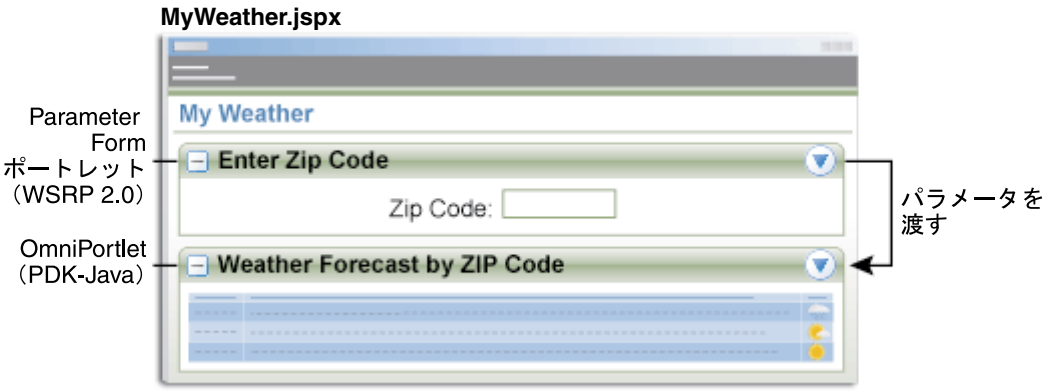
6

ポートレット間の通信

このレッスンでは、2 つのポートレットをページに追加し、一方のポートレットによってもう一方のポートレットのコンテンツが制御されるように構成する方法を学びます。ポートレット間の通信を例示するために、Parameter Form ポートレット（サンプルの WSRP 2.0 ポートレットの 1 つ）および OmniPortlet のインスタンス（Oracle PDK ポートレット）を使用します。どちらのポートレットも、第 3 章「最初のポートレットの構築およびテスト」でインストールした Preconfigured OC4J を介して使用可能です。

図 6-1 に、このレッスンを終えた時点での（MyWeather.jspx という名前の）ページの外観を示します。

図 6-1 レッスン 6 を終えた時点での MyWeather.jspx



概要

次の手順で、2 つのポートレット間の通信を行います。

- 手順 1: ポートレット・プロデューサの登録
- 手順 2: ページへの Parameter Form ポートレットの配置
- 手順 3: Parameter Form ポートレットのカスタマイズ
- 手順 4: ページへの OmniPortlet の配置
- 手順 5: Web サービスを使用する OmniPortlet の構築
- 手順 6: 両ポートレットの構成
- 手順 7: ポートレットの相互作用のテスト

前提条件

このレッスンで使用するポートレットはどちらも、（第 3 章の「手順 2: 接続の設定」でインストールおよび初期化した）WebCenter Preconfigured OC4J を介して使用可能です。この OC4J インスタンスがもう実行されていない場合は、このレッスンを開始する前に、JDeveloper ツールバーの一番右にある**「WebCenter Preconfigured OC4J の起動」**をクリックします。

手順 1: ポートレット・プロデューサの登録

ポートレットをページに追加する前に、ポートレットのプロデューサを WebCenter アプリケーションに登録する必要があります。このレッスンでは、次のポートレットのプロデューサを登録する必要があります。

- Parameter Form ポートレット（WSRP ポートレット・プロデューサ）
- OmniPortlet（Oracle PDK ポートレット・プロデューサ）

まずは、Parameter Form ポートレットから開始します。

1. Parameter Form ポートレット・プロデューサが稼働していることを確認します。ブラウザで、次のプロデューサ・テスト・ページ URL を入力します。

   `http://localhost:6688/portletapp/info`

   プロデューサが稼働中の場合は、WSRP プロデューサ・テスト・ページに 2 つの WSRP プロデューサ登録 URL（WSDL ドキュメント）が表示されます。この WebCenter アプリケーションでは、**WSRP v2 WSDL** の URL を使用します。

   `http://localhost:6688/portletapp/portlets/wsrp2?WSDL`

   次に、プロデューサをアプリケーションに登録します。

2. アプリケーション・ナビゲータで、**「Portlet Producers」**を右クリックし、**「新規 WSRP プロデューサ」**をクリックします。
3. WSRP ポートレット・プロデューサの登録ウィザードが表示されたら、**「次へ」**をクリックして、「ようこそ」ページの先に進みます。
4. 「名前」フィールドに `SampleWSRPPortletsProducer` を入力します。
5. **「次へ」**をクリックします。
6. 「接続」ページで、プロデューサの**「URL エンドポイント」**を入力します。

   WebCenter Preconfigured OC4J を介して Parameter Form ポートレットを生成するには、次の URL を入力します。

   `http://localhost:6688/portletapp/portlets/wsrp2?WSDL`

   このチュートリアルでは、URL 内に `localhost` を使用すると、Preconfigured OC4J がインストールされているローカル・コンピュータを参照できることを前提としています。（そ

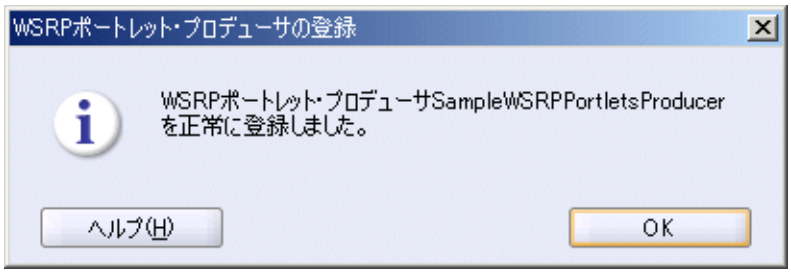
うでない場合は、localhost をご使用のコンピュータの IP アドレスに置き換えてください。）

ローカルの Preconfigured OC4J を使用しているため、プロキシ情報は必要ありません。**「プロキシを使用」**オプションが選択されていないことを確認してください。

7. **「次へ」**をクリックします。
8. 「登録の詳細」ページで、デフォルトのタイムアウト値の 30 秒を受け入れ、**「終了」**をクリックします。このチュートリアルでは、詳細オプションを設定する必要はありません。

   次のメッセージが表示されます（図 6-2）。

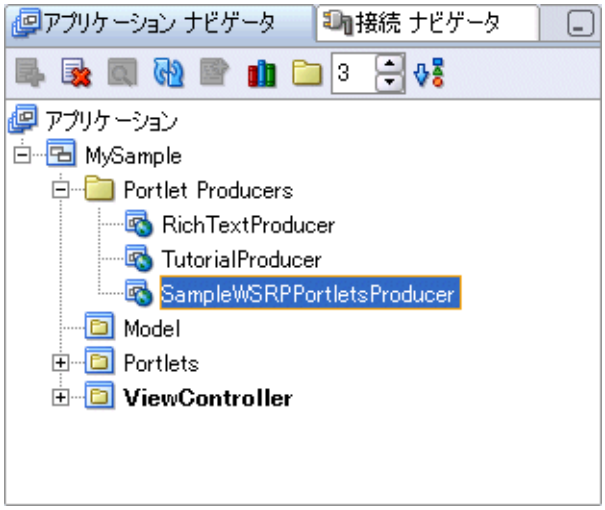
図 6-2 WSRP ポートレット・プロデューサ登録の正常完了



9. **「OK」**をクリックして、メッセージ・ボックスを閉じます。

   アプリケーション・ナビゲータで、「Portlet Producers」セクションの下に、新しいプロデューサが表示されます（図 6-3）。

図 6-3 正常に登録されたサンプル WSRP ポートレットのプロデューサ



「表示」メニューからコンポーネント・パレットを開き、そのドロップダウン・リストを開きます。このリストから、**「SampleWSRPPortletsProducer」**を見つけて選択します。複数のポートレットが選択可能になっています。その中に Parameter Form ポートレットがあります。このレッスンでは、このポートレットを使用します。（コンポーネント・パレットに何も表示されない場合は、「設計」ビューで MyPage.jspx などの Java Server Faces ページを表示してみてください。これによって、すべてのポートレットが表示されます。）

Parameter Form ポートレットについては、後で再び使用します。まずは、OmniPortlet プロデューサを登録する必要があります。

10. 最初に、OmniPortlet プロデューサが稼働していることを確認します。ブラウザで、次のプロデューサ・テスト・ページ URL を入力します。

    `http://localhost:6688/portalTools/omniPortlet/providers/omniPortlet`

    プロデューサが使用可能な場合は、OmniPortle のプロデューサ・テスト・ページが表示されます。

    次に、OmniPortlet プロデューサを登録します。このチュートリアルでは、今までに WSRP プロデューサのみを登録しました。OmniPortlet は Oracle PDK ポートレットなので、このプロデューサの登録には別のウィザードを使用します。

11. アプリケーション・ナビゲータで、**「Portlet Producers」**を右クリックし、**「新規 Oracle PDK プロデューサ」**をクリックします。

12. PDK ポートレット・プロデューサの登録ウィザードが表示されたら、**「次へ」**をクリックします。

13. 「名前」フィールドに `OmniPortletProducer` を入力します。

14. **「次へ」**をクリックします。

15. 「接続」ページで、プロデューサの**「URL エンドポイント」**を入力します。

    WebCenter Preconfigured OC4J を介して OmniPortlet を生成するには、次の URL を入力します。

    `http://localhost:6688/portalTools/omniPortlet/providers/omniPortlet`

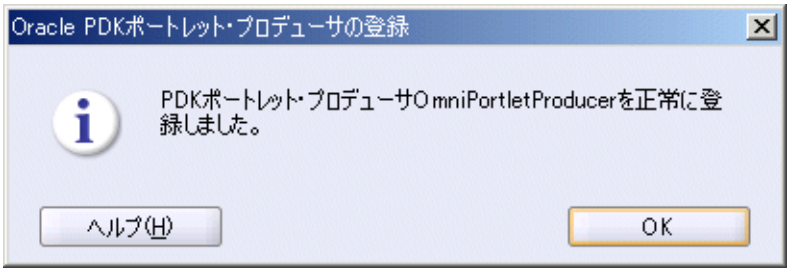
    このチュートリアルでは、URL 内に `localhost` を使用すると、Preconfigured OC4J がインストールされているローカル・コンピュータを参照できることを前提としています。(そうでない場合は、`localhost` をご使用のコンピュータの IP アドレスに置き換えてください。)

    「プロキシを使用」オプションが選択されていないことを確認してください。

16. **「終了」**をクリックします。

    次のメッセージが表示されます(図 6-4)。

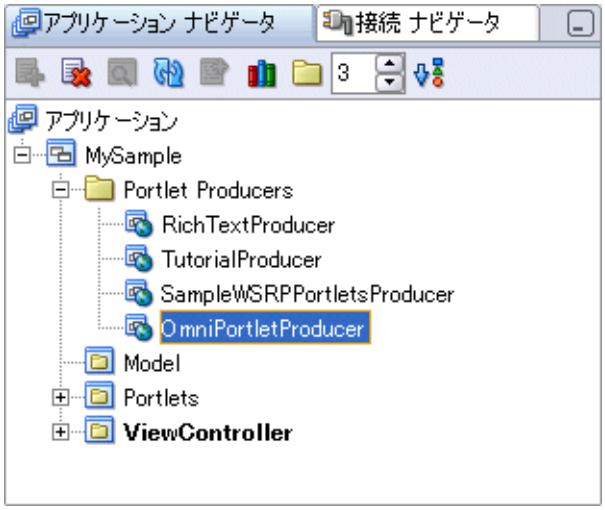
図 6-4 Oracle PDK ポートレット・プロデューサ登録の正常完了



17. 「**OK**」をクリックして、メッセージ・ボックスを閉じます。

    アプリケーション・ナビゲータで、「Portlet Producers」セクションの下に、両方のプロデューサが表示されます（図 6-5）。

図 6-5 正常に登録された OmniPortlet のプロデューサ



18. コンポーネント・パレットを再度開き、ドロップダウン・リストをクリックして、「OmniPortletProducer」が表示されていることを確認します。このリストから「**OmniPortletProducer**」を選択します。

    複数のポートレットが選択可能になっています。その中に OmniPortlet があります。このレッスンでは、このポートレットを使用します。

両方のプロデューサがチュートリアル・アプリケーションに登録されたので、それらのポートレットをページにドロップできるようになります。

手順 2: ページへの Parameter Form ポートレットの配置

この手順では、Parameter Form ポートレットをページ上に配置し、パブリック・ポートレット・パラメータを持つポートレットが JDeveloper でどのように処理されるかを確認します。次にページを実行し、ポートレットのカスタマイズを介して、ユーザーに ZIP コードの入力を促すポートレットを作成します。このレッスンの以降の部分で、パラメータとしての ZIP コードを OmniPortlet に渡す方法を学びます。OmniPortlet では、このパラメータを使用して気象ポートレットのコンテンツが制御されます。

まずは、MyWeather.jspx という名前の新しいページを作成します。このページを使用して、ポートレット間の通信について学びます。

1. MyWeather.jspx という名前の新しい Java Server Faces ページを作成します。
   a. アプリケーション・ナビゲータで、「**ViewController**」を右クリックし、「**新規**」を選択します。
   b. 「カテゴリ」ペインで、「Web Tier」の下の「**JSF**」を選択します。
   c. 「項目」で「**JSF JSP**」を選択し、「**OK**」をクリックします。
   d. 「**次へ**」をクリックして、「ようこそ」ページをスキップします。
   e. 「ファイル名」フィールドに `MyWeather` を入力します。
   f. 「タイプ」で「**JSP ドキュメント**」をクリックし、「**次へ**」をクリックします。
   g. 「**新規マネージド Bean での UI コンポーネントの自動公開**」を選択し、「**次へ**」をクリックします。

h. 「タグ・ライブラリ」ページで、次のライブラリが**「選択済のライブラリ」**ペインに表示されていることを確認します。

ADF Faces Components 10_1_3_2_0

ADF Faces HTML 10_1_3_2_0

ADF Portlet Components 10_1_3_2_0

Customizable Components Core 10_1_3_2

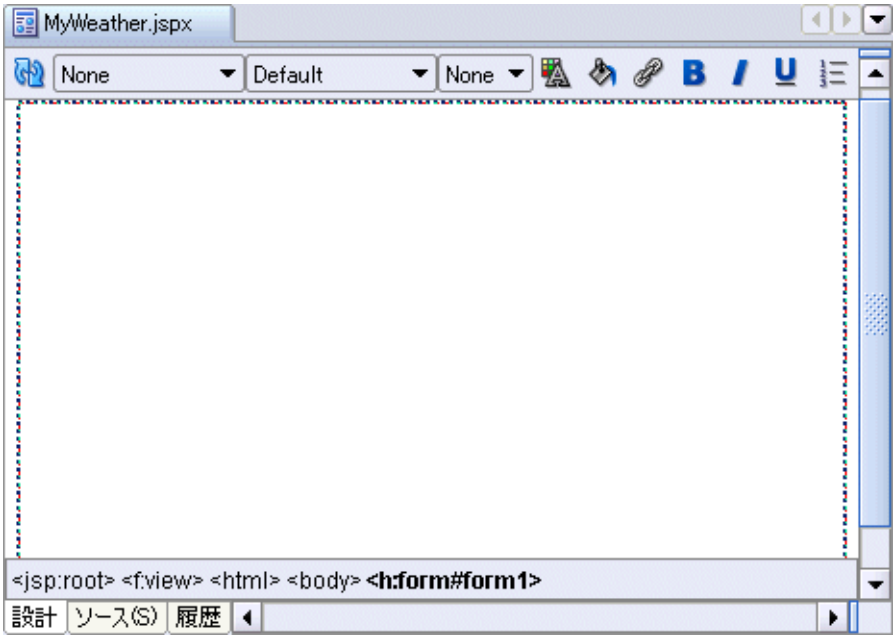
JSF Core 1.0

JSF HTML 1.0

i. その他のオプションは設定する必要がないので、このページで**「終了」**をクリックします。

ビジュアル・エディタに MyWeather.jspx が開かれ（図 6-6）、ポートレットの追加を開始できるようになります。

図 6-6 空の MyWeather.jspx

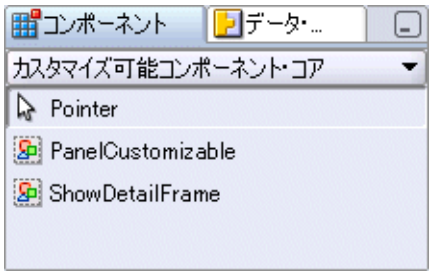


2. panelCustomizable コンポーネントをページに追加します。このコンポーネントを使用して、両方のポートレット（Parameter Form ポートレットと OmniPortlet）を格納し、水平または垂直レイアウトを指定します。

   a. コンポーネント・パレットを開きます。

   b. 「**カスタマイズ可能コンポーネント・コア**」のリストにスクロールします（図 6-7）。

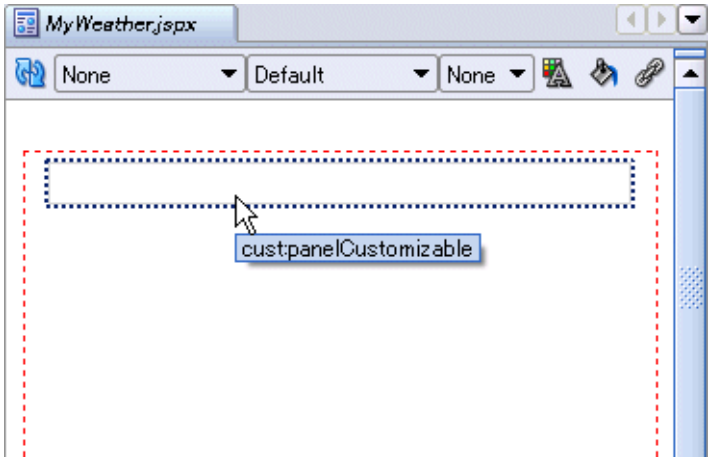
図 6-7 コンポーネント・パレット - カスタマイズ可能コンポーネント・コア



   c. 「**PanelCustomizable**」を選択し、構造ウィンドウ内の「h:form」タグの上にドラッグします。

   ページの一番上に、panelCustomizable コンポーネントが表示されます（図 6-8）。

図 6-8 MyWeather.jspx - PanelCustomizable コンポーネント



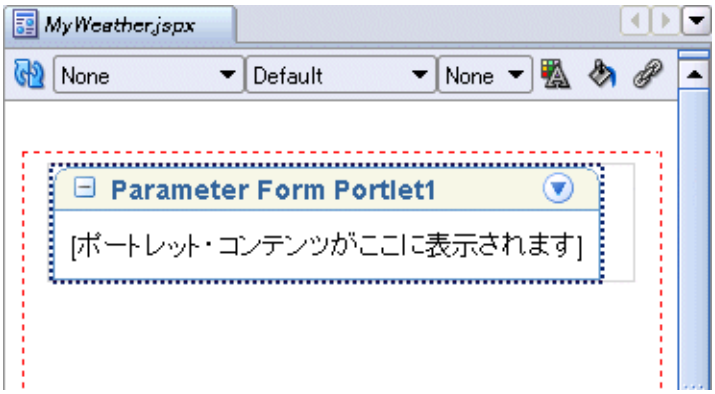
   panelCustomizable コンポーネントは、ページの h:form 内に配置する必要があります。構造ウィンドウ内を調べて、フレームがフォームの中に配置されていることを確認します。ポートレットにはカスタマイズ機能（ページ上でポートレットの最大化 / 最小化や位置変更を行う機能）が自動的に備わっているため、showDetailFrame を追加する必要はないことを思い出してください。

3. いよいよ、Parameter Form ポートレットを配置します。コンポーネント・パレットから、「**SampleWSRPPortletsProducer**」を選択します。

4. **「Parameter Form ポートレット」**を選択し、構造ウィンドウ内の「`cust:panelCustomizable`」コンポーネントの上にドラッグ・アンド・ドロップします。

   ビジュアル・エディタは、図 6-9 のように表示されます。

図 6-9 MyWeather.jspx - panelCustomizable 内の Parameter Form ポートレット

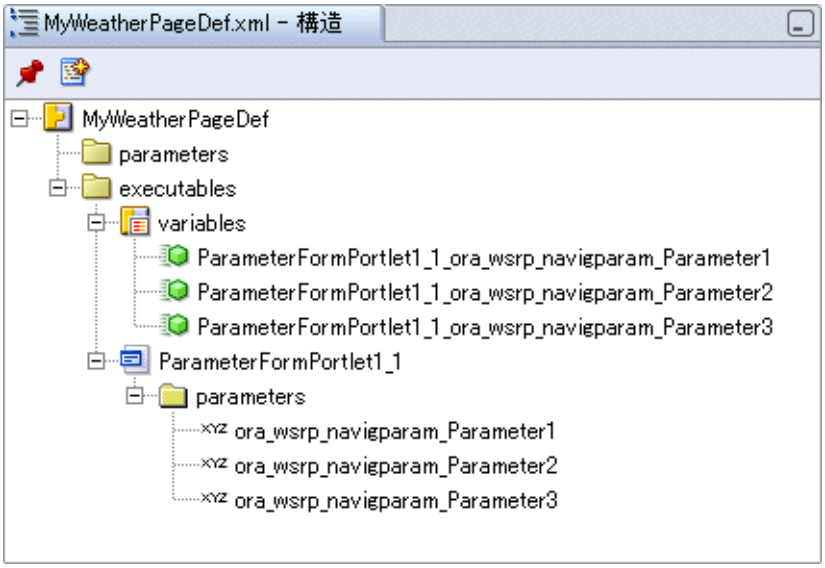


ページを実行する前に、基礎となるページ定義を調べて、このようなビルトイン・パラメータを備えたポートレットに JDeveloper がどのように応答するかを確認します。

アプリケーション・ナビゲータで、**「MyWeather.jspx」**を右クリックし、**「ページ定義に移動」**を選択します。ページ定義がまだ存在していない場合は、**「はい」**をクリックします。

構造ウィンドウ（図 6-10）には、Parameter Form の各ポートレット・パラメータに対して 1 つずつエントリが表示されます（`ora_wsrp_navigparam_Parameter1`、`ora_wsrp_navigparam_Parameter2` および `ora_wsrp_navigparam_Parameter3`）。また、3 つのページ変数エントリと、それに対応する名前および接頭辞 <portletname_n_> も表示されます（この場合は `ParameterFormPortlet1_1_`）。

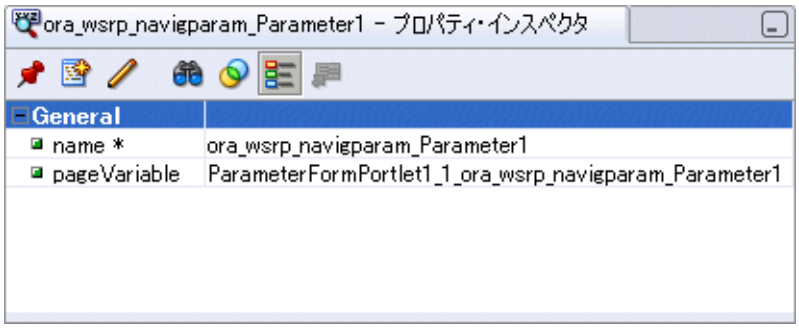
図 6-10 MyWeatherPageDef.xml - Parameter Form ポートレット・パラメータ



ポートレットをページ上にドロップすると、JDeveloper によって、ポートレットがパブリック・ポートレット・パラメータを持つかどうかが確認され、持つ場合は、対応する数のページ変数が自動的に追加されてポートレット・パラメータにマップされます。

プロパティ・インスペクタを使用して、マッピングの 1 つを調べます。たとえば、`ora_wsrp_navigparam_Parameter1` が、`ParameterFormPortlet1_1_ora_wsrp_navigparam_Parameter1` という名前のページ変数にマップされています（図 6-11）。

図 6-11 プロパティ・インスペクタ - ページ変数にマップされたポートレット・パラメータ

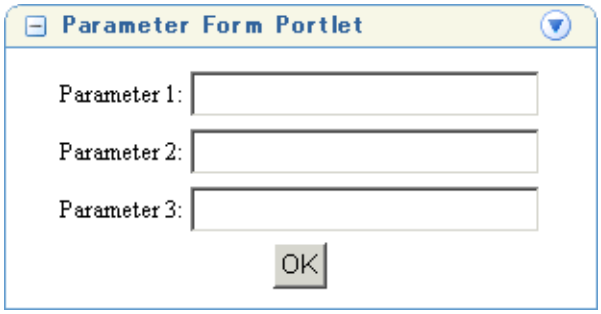


今度は、ページを実行して、デフォルトの Parameter Form ポートレットがどのように表示されるかを確認します。

5. まず、変更をすべて保存します。JDeveloper のツールバーで、**「すべて保存」**アイコンをクリックします。
6. アプリケーション・ナビゲータで、**「MyWeather.jspx」**を右クリックし、**「実行」**を選択します。

   ブラウザに、デフォルトの Parameter Form ポートレットが表示されます（図 6-12）。

図 6-12 デフォルトの Parameter Form ポートレット



ポートレットの最初のバージョンではデフォルト・テキストが表示されることがわかります。次の手順では、ユーザーに US ZIP コードの入力を促すようにポートレットをカスタマイズします。このレッスンの目的は、（ZIP コードなどの）パラメータ値を受け入れて別のポートレットに渡すようにこのポートレットを設定することなので、忘れないでください。

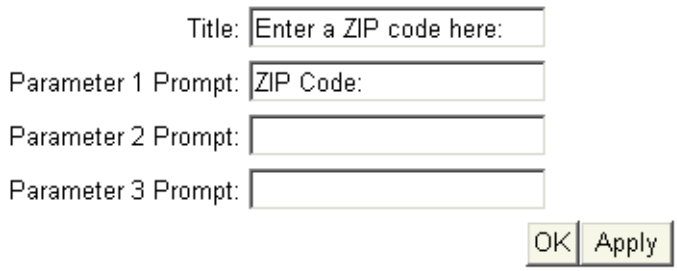
手順 3: Parameter Form ポートレットのカスタマイズ

この手順では、タイトルおよびプロンプトを変更することにより、Parameter Form ポートレットをカスタマイズします。また、不要なパラメータを非表示にします。Parameter Form ポートレットには 3 つのパラメータが用意されていますが、このチュートリアルで必要なのは、ポートレット間で受渡しされるパラメータを示すための 1 つのみです。

1. ブラウザでポートレットを表示し、ポートレット・ヘッダーの**アクション**・アイコンをクリックします。
2. メニュー・オプション**「カスタマイズ」**をクリックします。
3. 「タイトル」を `Enter a ZIP code here:` に変更します。
4. 「パラメータ 1 のプロンプト」には `ZIP Code:` を入力します。
5. 2 つ目と 3 つ目のパラメータは必要ないので、非表示にしてかまいません。これを行うには、空白を残さないように注意して、デフォルトのプロンプト・テキストを削除します。

   「カスタマイズ」オプションは、図 6-13 のように表示されます。

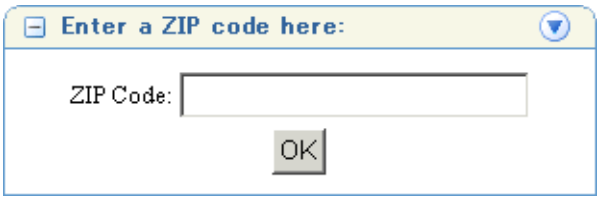
図 6-13 Parameter Form ポートレットのカスタマイズ・オプション



6. **「OK」**をクリックします。

ポートレットは、ブラウザでは図 6-14 のように表示されます。

図 6-14 カスタマイズ済の Parameter Form ポートレット



次の手順に進む前に、これまでに行った内容を振り返ってみます。最初に、MyWeather.jspx という名前のページを作成し、1 つの Parameter Form ポートレットを追加しました。次に、ポートレットのカスタマイズを介して、US ZIP コードを受け入れるようにパラメータの 1 つ（`ora_wsrp_navigparam_Parameter1`）を設定しました。JDeveloper によって、この特定のパラメータが `ParameterFormPortlet1_1_ora_wsrp_navigparam_Parameter1` という名前のページ変数に自動的にマップされました。後で、この変数を使用して、ZIP コードを別のポートレットに渡します。

しかしまずは、2 つ目のポートレット OmniPortlet を追加します。

手順 4: ページへの OmniPortlet の配置

この手順では、Parameter Form ポートレットの下に OmniPortlet を配置します。

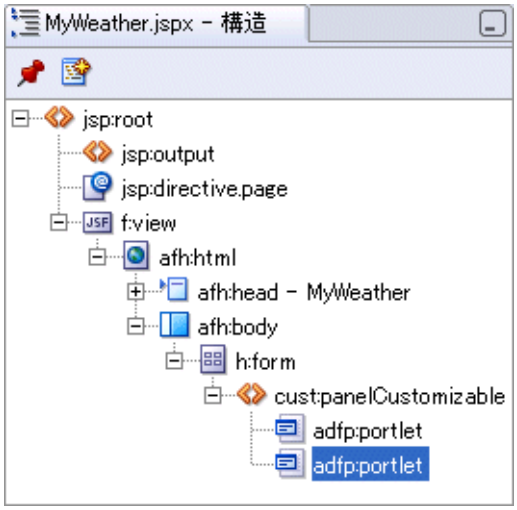
1. JDeveloper で、MyWeather.jspx を表示します。アプリケーション・ナビゲータで、「**MyWeather.jspx**」を右クリックし、「**開く**」をクリックします。
2. コンポーネント・パレットを開き、「**OmniPortletProducer**」を見つけます。
3. 「**OmniPortletProducer**」を選択します。
4. 「**OmniPortlet**」を選択し、構造ウィンドウ内の「`cust:PanelCustomizable`」の上にドラッグします。

   ビジュアル・エディタは、図 6-15 のように表示されます。

図 6-15 MyWeather.jspx - PanelCustomizable に追加された OmniPortlet

構造ウィンドウは、図 6-16 のように表示されます。

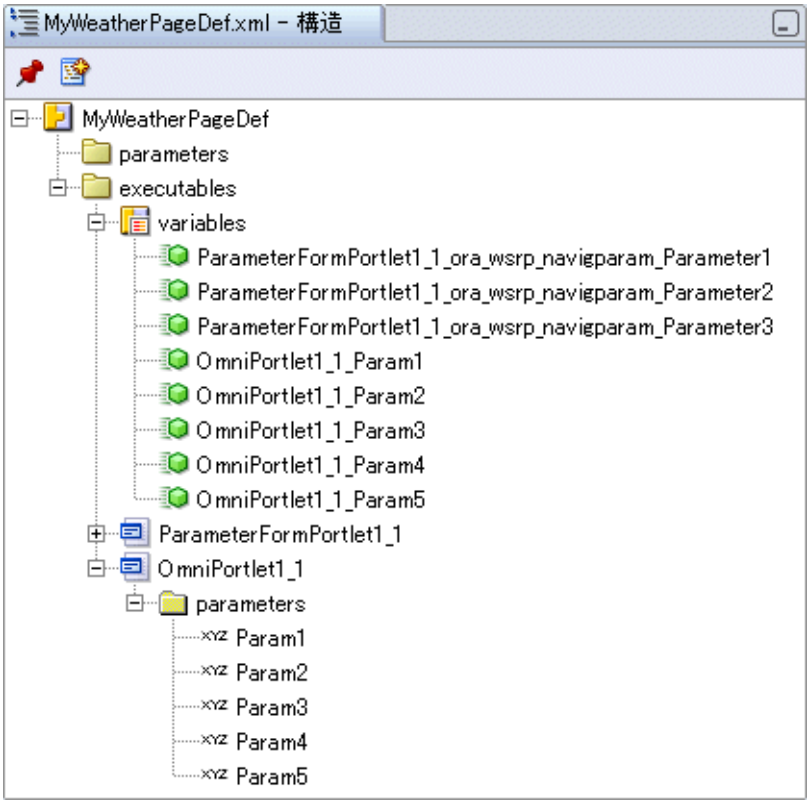
図 6-16 構造ウィンドウ - 新しい OmniPortlet の表示



5. ここで、基礎となるページ定義をもう一度見てみます。アプリケーション・ナビゲータで、**「MyWeather.jspx」**を右クリックし、**「ページ定義に移動」**を選択します。

   構造ウィンドウ（図 6-17）に、OmniPortlet の新しいエントリが表示されます。このポートレットには 5 つのポートレット・パラメータ（`Param1` ～ `Param5`）があります。これらもすでに、ページ変数（`OmniPortlet1_1_Param1` ～ `OmniPortlet1_1_Param5`）に自動的にマップされています。

図 6-17 MyWeatherPageDef.xml - OmniPortlet ポートレット・パラメータ



「手順 6: 両ポートレットの構成」で、これらのポートレット・パラメータを使用してポートレット間通信を構成します。しかしまずは、OmniPortlet ポートレットの外観を確認します。

6. まず、変更をすべて保存します。JDeveloper のツールバーで、**「すべて保存」**アイコンをクリックします。

7. 再度ページを実行する前に、埋込み OC4J サーバーを停止します。メイン・メニューから**「実行」**、**「終了」**、**「埋込み OC4J サーバー」**の順に選択します。

   あるいは、埋込み OC4J サーバーのログ・ウィンドウで、赤い四角形の「終了」アイコンをクリックします。

8. アプリケーション・ナビゲータで、「**MyWeather.jspx**」を右クリックし、「**実行**」を選択します。

   ブラウザで、Parameter Form ポートレットの下に OmniPortlet が表示されます（図 6-18）。

図 6-18 デフォルト（空白）の OmniPortlet

ポートレットの初期バージョンは空白であることがわかります。次の手順では、OmniPortlet を使用して、Web サービスに基づいた気象ポートレットを構築します。

手順 5: Web サービスを使用する OmniPortlet の構築

OmniPortlet は最も用途の広いポートレットの 1 つであり、様々なレイアウトを使用して様々なデータソースからデータを公開できます。この手順では、Web サービスに基づいた気象ポートレットを構築します。

注意： ファイアウォールの内側で作業している場合、気象 Web サービスへのアクセスには追加構成がいくつか必要です。特に、HTTP プロキシの詳細を Omniportlet プロデューサの provider.xml ファイルの `<proxyInfo>` タグに追加する必要があります。このファイルは次の場所にあります。

JDEV_HOME¥jdev¥extensions¥oracle.adfp.seededoc4j.10.1.3.2.¥j2ee¥home¥applications¥portalTools¥omniPortlet¥WEB-INF¥providers¥omniPortlet

`provider.xml` ファイルに変更を加えた場合、ポートレット・プロデューサを WebCenter アプリケーション側でリフレッシュする必要があります。プロキシの詳細を更新し、ポートレット・プロデューサをリフレッシュするには、次の手順を実行します。

1. WebCenter アプリケーション（コンシューマ）を停止します。
2. `provider.xml` ファイルを更新し、変更内容を保存します。
3. Oracle JDeveloper で、ポートレット・プロデューサを WebCenter アプリケーション（コンシューマ）側でリフレッシュします。
4. WebCenter アプリケーションを再デプロイします。

詳細は、『Oracle WebCenter Framework 開発者ガイド』のファイアウォールの外側のデータにアクセスするための OmniPortlet プロデューサの構成に関する項を参照してください。

---

1. ブラウザで、MyWeather.jspx を表示します。アプリケーション・ナビゲータで、**「MyWeather.jspx」**を右クリックし、**「実行」**を選択します。
2. ブラウザで**「定義」**リンクをクリックして、OmniPortlet ウィザードを起動します。
3. **「Web サービス」**を選択し、**「次へ」**をクリックします。

   このチュートリアルの完了後、スプレットシート（文字区切りの値）や XML、既存の Web ページからのアプリケーション・データといった他のデータソースを OmniPortlet で使用する方法を学ぶこともできます。これらのオプションについてはすべて、『Oracle WebCenter Framework 開発者ガイド』で説明されています。
4. 次の場所にある Oracle Technology Network から利用可能な、デモ用の気象 Web サービスの **WSDL URL** を入力します。

   ```
   http://webservices.oracle.com/WeatherWS/WeatherWS?WSDL
   ```

   この Web サービスは 1 つのメソッド（WeatherWS.giveMeSomeWeatherInfo）を持ち、1 つのパラメータ（param0）を受け入れます。param0 を介して有効な US ZIP コードを渡すと、Web サービスは、指定された地域の気象情報を返します。

   注意 : この Web サービスは、実際の気象情報を返しません。デモ目的でのみ使用できます。
5. このメソッド・パラメータにアクセスするには、**「メソッドを表示」**をクリックします。

   「param0」というラベルの付いたパラメータがページに表示されます。
6. 「param0」パラメータ・フィールドに ##Param1## を入力します。

   これによって、Web サービスが Omniportlet パラメータ Param1 からの値を受け入れるように設定されます。後で、ポートレット・パラメータ Param1 を、ParameterFormPortlet1_1_ora_wsrp_navigparam_Parameter1 という名前のページ変数にマップします（「手順 6: 両ポートレットの構成」を参照）。
7. 「ポートレット・パラメータ」で、「Param1」にデフォルト値を設定します。94065 を入力します。

   これによって、Web サービスがデフォルトで ZIP コード 94065（カリフォルニア州レッドウッド市）の気象情報を返すように設定されます。
8. **「次へ」**をクリックします。フィルタ・オプションを設定する必要はないので、再度**「次へ」**をクリックします。
9. 「表示」ページで、表 6-1 に示す値を入力します。

表 6-1 OmniPortlet の「表示」ページの設定

設定	値
タイトル	ポートレットのタイトルとして Weather Forecast を入力します。
ヘッダー・テキスト	ポートレット・ヘッダーに Param1（現在の ZIP コード）の現在値を表示します。次のように入力します。 For ZIP Code ##Param1##

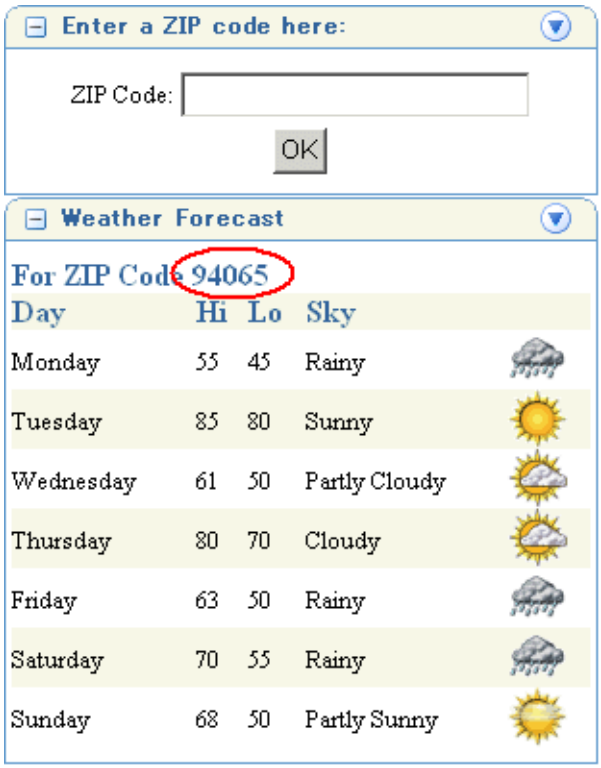
10. **「次へ」**をクリックし、気象データの列ラベル、列および表示形式のプロパティを指定します（表 6-2 を参照）。

表 6-2 気象予報ポートレットの列プロパティ

名前	列ラベル	列	表示形式
Field1	Day	dayOfWeek	テキスト
Field2	Hi	hiTemp	テキスト
Field3	Lo	lowTemp	テキスト
Field4	Sky	sky	テキスト
Field5	（空白）	img	イメージ

11. **「終了」**をクリックして、レッドウッド市の気象情報を表示します。ブラウザに、両方のポートレットが表示されます（図 6-19）。

図 6-19 気象 Web サービスを表示する OmniPortlet



Omniportlet では、ZIP コードが 94065 であるカリフォルニア州レッドウッド市の気象予報が表示されます。このデータを提供する Web サービスは、有効な UI ZIP コードが渡されれば、他の地域の気象情報を返すこともできます。次の手順では、この 2 つのポートレットをリンクし、最初のポートレット（Parameter Form ポートレット）に OmniPortlet のコンテンツを制御させる方法を学びます。

手順 6: 両ポートレットの構成

この手順では、2 つのポートレットのポートレット間通信を有効にします。

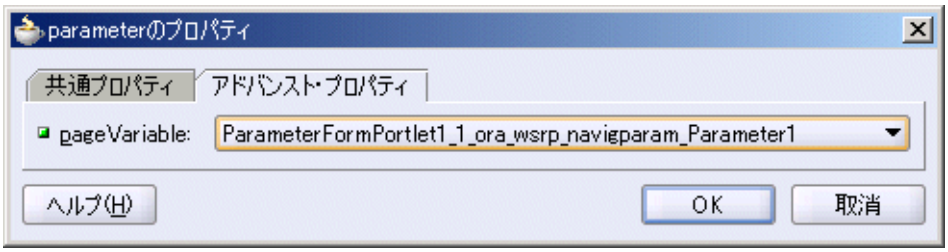
1. JDeveloper で、MyWeather.jspx のページ定義を表示します。アプリケーション・ナビゲータで、**「MyWeather.jspx」**を右クリックし、**「ページ定義に移動」**を選択します。
2. 構造ウィンドウを使用して、MyWeatherPageDef.xml を調べます。「実行可能ファイル」セクションには、次のものが表示されます。
   - ParameterFormPortlet1_1: 3 つのポートレット・パラメータ（`ora_wsrp_navigparam_Parameter1`、`ora_wsrp_navigparam_Parameter2` および `ora_wsrp_navigparam_Parameter3`）があります。
   - OmniPortlet1_1: 5 つのポートレット・パラメータ（`Param1` ～ `Param5`）があります。
   - 変数 : 8 つのページ変数がそれぞれ別々のポートレット・パラメータにマップされます。（パラメータ付きの）ポートレットがページ上に配置されると、JDeveloper によってこれらのマッピングが自動的に作成されます。

   各ポートレットでは複数のパラメータをサポートしていますが、ここでは、ポートレット間通信を例示するため、各ポートレットから 1 つのパラメータしか使用しません。
   - `ora_wsrp_navigparam_Parameter1`: このパラメータを介して、Parameter Form ポートレットが ZIP コードを受け入れるように設定しました（「手順 3: Parameter Form ポートレットのカスタマイズ」）。
   - `Param1`: このパラメータを介して、OmniPortlet が ZIP コードを受け入れるように構成しました（「手順 5: Web サービスを使用する OmniPortlet の構築」）。

   次の手順では、この 2 つのパラメータを同じページ変数にマップします。その際に、2 つのポートレット間のパラメータの受渡しを促進します。現時点で `ora_wsrp_navigparam_Parameter1` はページ変数 `ParameterFormPortlet1_1_ora_wsrp_navigparam_Parameter1` にマップされているので、今度は `Param1` もこのページ変数にマップします。
3. 構造ウィンドウを使用して、OmniPortlet と Parameter Form ポートレットをリンクします。
   a. **「Param1」**を右クリックし、**「プロパティ」**を選択します。

      「Param1」は、「MyWeatherPageDef¥executables¥OmniPortlet1_1¥parameters」の下にあります。
   b. **「アドバンスト・プロパティ」**をクリックします。
   c. 「pageVariable」ドロップダウン・リストから、**「ParameterFormPortlet1_1_ora_wsrp_navigparam_Parameter1」**を選択します（図 6-20）。

図 6-20 Parameter Form ポートレットと同じページ変数にマップされた OmniPortlet Param1



   d. **「OK」**をクリックします。
4. JDeveloper のツールバーで、**「すべて保存」**をクリックします。

今度は、ページを実行して、ポートレットがともに動作することを確認します。

手順 7: ポートレットの相互作用のテスト

これから、MyWeather.jspx のポートレットがともに動作することを確認します。

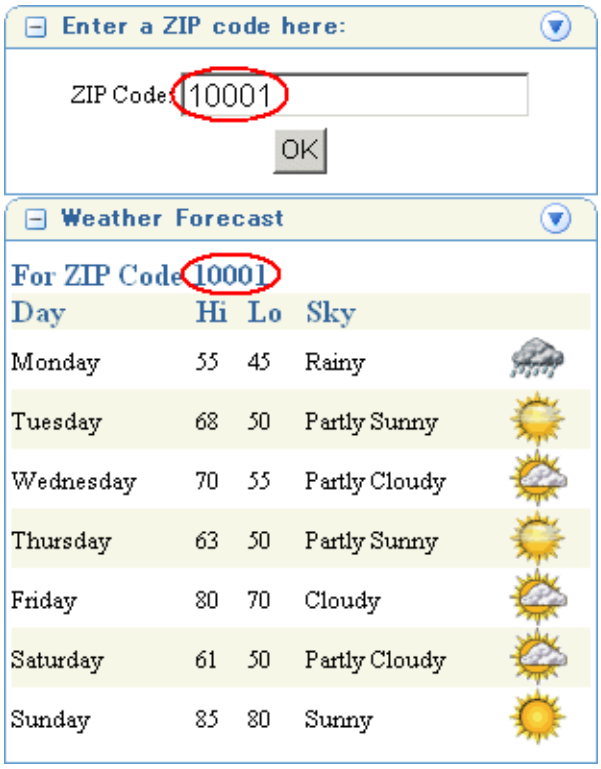
1. 再度ページを実行する前に、埋込み OC4J サーバーを停止します。メイン・メニューから**「実行」**、**「終了」**、**「埋込み OC4J サーバー」**の順に選択します。
2. アプリケーション・ナビゲータで、**「MyWeather.jspx」**を右クリックし、**「実行」**を選択します。

   コンパイル・エラーがなければ、ページが新しいブラウザ・ウィンドウ内に開き、レッドウッド市地域（ZIP コード 94065）の気象予報が表示されます。
3. **「ZIP Code」**フィールドに `10001` を入力します。

   これはニューヨークの ZIP コードです。
4. **「OK」**をクリックします。

   気象予報ポートレットが変更され、（図 6-21 のような）ニューヨークの気象情報が表示されます。

図 6-21 ニューヨークの気象を表示するポートレット



おめでとうございます。これで、このレッスンは終わりです。2 つのポートレットが互いに通信できるようになりました。

次のレッスンでは、WebCenter アプリケーションのファイル・システム・コンテンツを公開する方法を学びます。

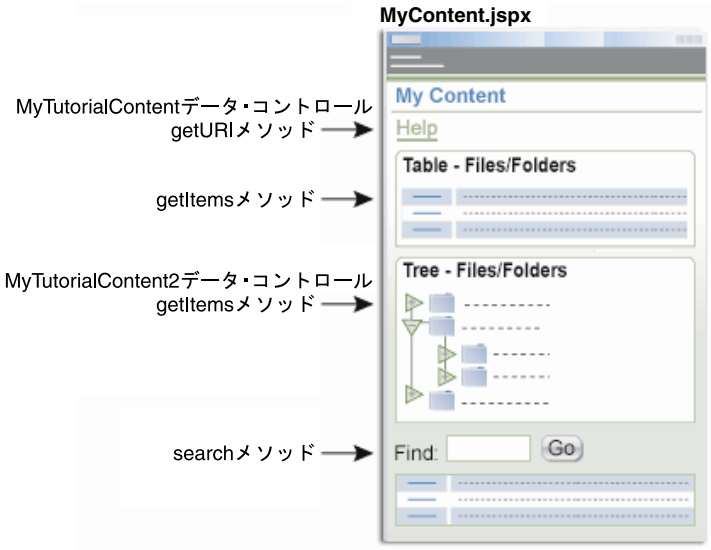
7

ページへのコンテンツの追加

このレッスンでは、WebCenter アプリケーションのファイル・システムに存在するコンテンツを公開する方法を学びます。Oracle JDeveloper を使用して、コンテキスト・リッチなアプリケーションを簡単に構築できることがわかります。また、ファイルおよびフォルダの情報を表示する様々な方法を試してみます。

図 7-1 に、この章を終えた時点でのページ（MyContent.jspx）の外観を示します。

図 7-1 レッスン 7 を終えた時点での MyContent.jspx



概要

次の手順で、チュートリアル・アプリケーションにコンテンツを追加します。

- 手順 1: データ・コントロールの作成
- 手順 2: 設計時のページへのコンテンツの追加
- 手順 3: 表でのフォルダ・コンテンツの表示
- 手順 4: ツリーでのフォルダ・コンテンツの表示
- 手順 5: フォルダ・コンテンツの検索

前提条件

このレッスンでは、第 2 章「チュートリアルを始める前に」でダウンロードしたサンプルにアクセスする必要があります。開始する前に、サンプル・コンテンツが格納される場所を書き留めておいてください（「サンプル・チュートリアル・ファイルのダウンロード」も参照）。

手順 1: データ・コントロールの作成

この手順では、ファイル・システムに格納されているサンプル・チュートリアル・コンテンツへのアクセスおよび公開を可能にするデータ・コントロールを定義します。データ・コントロールとは、アプリケーション内の UI コンポーネントの作成に使用されるすべてのデータ・オブジェクト、コレクション、メソッドおよび操作のコンテナです。

1. アプリケーション・ナビゲータで、**「Model」**を右クリックし、**「新規」**を選択します。
2. **「Business Tier」**ノードを開き、**「コンテンツ・リポジトリ」**を選択します。
3. **「コンテンツ・リポジトリのデータ・コントロール」**を選択し、**「OK」**をクリックしてウィザードを表示します。

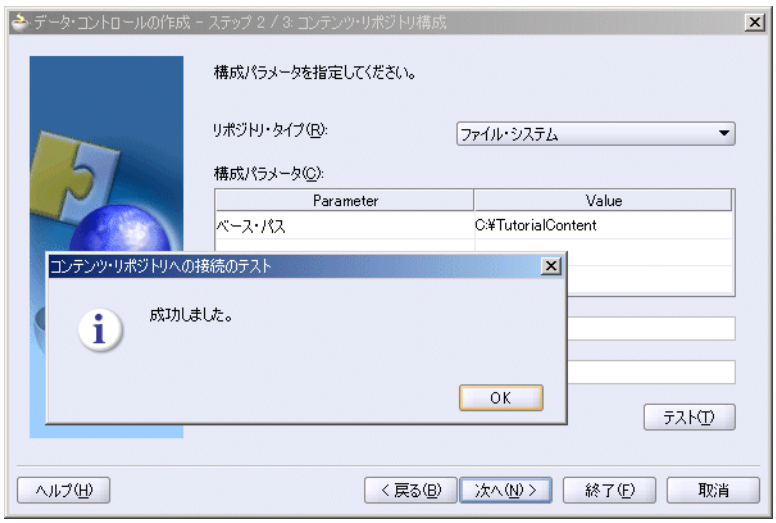
   このウィザードによって、コンテンツがファイル・システム上にある場合でも、コンテンツ・リポジトリのデータ・コントロールが作成されます。
4. **「次へ」**をクリックして、「ようこそ」ページをスキップします。
5. データ・コントロールに名前を付けます。`MyTutorialContent` と入力し、**「次へ」**をクリックします。
6. ここでは、ファイル・システム上のコンテンツを公開することにします。**「リポジトリ・タイプ」**ドロップダウン・リストをクリックし、**「ファイル・システム」**を選択します。

   また、Oracle Content DB リポジトリまたは OracleAS Portal リポジトリ内のコンテンツにアクセスするようにデータ・コントロールを設定することもできます。このようなタイプのデータ・コントロールの設定方法の詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。
7. 「ベース・パス」フィールドに、先に解凍したサンプル・コンテンツのパスを入力します。たとえば、`C:¥TutorialContent` などです。

   これによって、データ・コントロールにコンテンツの場所を認識させます。

8. 「テスト」ボタンをクリックして、接続詳細を適切に入力したかどうかを確認します。図 7-2 のような「成功しました。」というメッセージが表示されます。

図 7-2 ファイル・システムのデータ・コントロール - 接続のテスト



9. エラー・メッセージが表示された場合は、「OK」をクリックし、フルパスを指定するように注意して「ベース・パス」を編集します。テストが成功した場合は、「OK」をクリックしてメッセージ・ボックスを閉じます。

10. 「次へ」をクリックします。

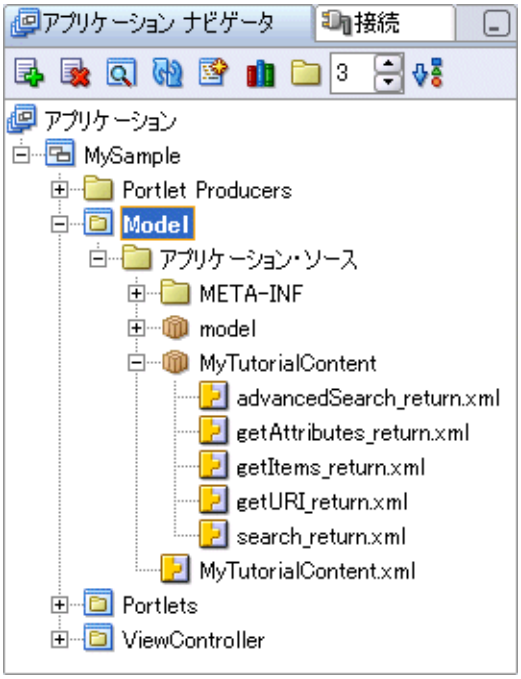
ファイル・システムのデータ・コントロールは、複数のデフォルト属性（name、path、URI、primaryType）と、オプションで 1 つのカスタム属性（lastModified）を公開します（図 7-3 を参照）。

図 7-3 ファイル・システムのデータ・コントロール - カスタム属性構成

11. デフォルト属性セットを受け入れるには、**「終了」**をクリックします。

    ここで、アプリケーション・ナビゲータを見てみます。「Model」、「アプリケーション・ソース」の下に、いくつかの新しいエントリが表示されています（図 7-4）。データ・コントロール、およびデータ・コントロールにより生成されるファイルの詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

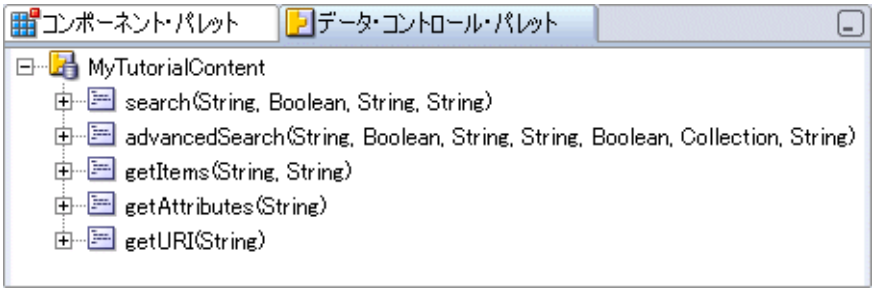
図 7-4 ファイル・システムのデータ・コントロール MyTutorialContent のファイル



12. データ・コントロール・パレットで新しいデータ・コントロールが使用可能になりますので、ここでそれを表示します。**「表示」**メニューから、**「データ・コントロール・パレット」**を選択します。

    「MyTutorialContent」の下に、新しいデータ・コントロールのオブジェクト、コレクション、メソッド、パラメータおよび操作の階層リストが表示されます（図 7-5）。

図 7-5 データ・コントロール・パレット - MyTutorialContent



ファイル・システムのデータ・コントロール（MyTutorialContent など）は、ファイルとフォルダの情報をアクセスおよび表示するための複数のメソッドを提供します。

- **search:** データ・コントロールを介して公開されたコンテンツに対して検索を実行できるようにします。
- **advancedSearch:** データ・コントロールを介して公開されたコンテンツに対して拡張検索を実行できるようにします。
- **getItems:** コンテンツ・リポジトリの特定の場所に格納されているファイルおよびフォルダを返します。
- **getAttributes:** 特定のファイルまたはフォルダの属性とその値のリストを返します。
- **getURI:** ファイルの URI を返します。このリリースでは、URI を介したフォルダへのダイレクト・アクセスはサポートされていません。

これらのメソッドの詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

これから、これらのオブジェクトのいくつかを使用する方法を示します。

手順 2: 設計時のページへのコンテンツの追加

この手順では、データ・コントロール・メソッド `getURI` を使用してファイルへのハイパーリンクを公開する方法を学びます。まずは、MyContent.jspx という名前の新しいページを作成します。このページ上に、サンプル・ファイルの 1 つ（help.html）へのリンクを追加します。

1. MyContent.jspx という名前の新しい Java Server Faces ページを作成します。
   a. アプリケーション・ナビゲータで、**「ViewController」**を右クリックし、**「新規」**を選択します。
   b. 「カテゴリ」ペインで、「Web Tier」の下の**「JSF」**を選択します。
   c. 「項目」で**「JSF JSP」**を選択し、**「OK」**をクリックします。
   d. **「次へ」**をクリックして、「ようこそ」ページをスキップします。
   e. 「UI」フィールドに `MyContent` を入力します。
   f. 「タイプ」で**「JSP ドキュメント」**をクリックし、**「次へ」**をクリックします。
   g. **「新規マネージド Bean での UI コンポーネントの自動公開」**を選択し、**「次へ」**をクリックします。
   h. 「タグ・ライブラリ」ページで、次のライブラリが**「選択済のライブラリ」**ペインに表示されていることを確認します。

      ADF Faces Components 10_1_3_2_0

      ADF Faces HTML 10_1_3_2_0

      ADF Portlet Components 10_1_3_2_0

      Customizable Components Core 10_1_3_2

      JSF Core 1.0

      JSF HTML 1.0

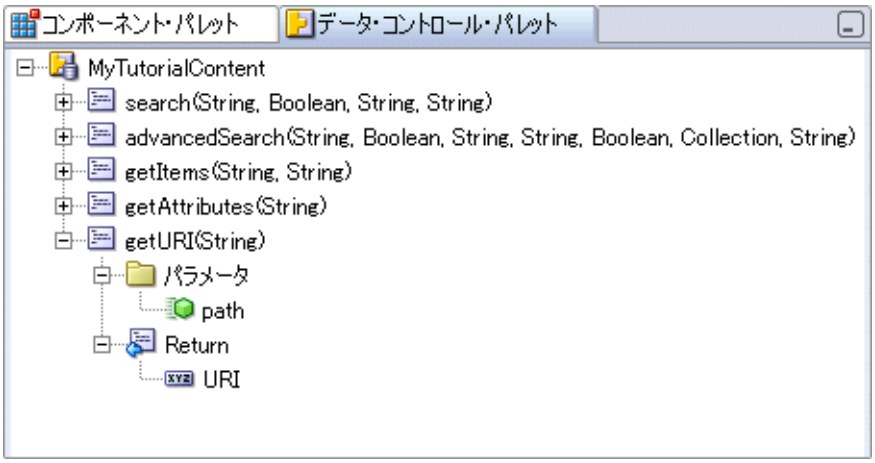
   i. その他のオプションは設定する必要がないので、このページで**「終了」**をクリックします。

ビジュアル・エディタに MyContent.jspx が開かれ、ファイル・システムからのコンテンツの追加を開始できるようになります。（ページが表示されない場合は、**「設計」**タブをクリックします。）

2. データ・コントロール・パレットで、「MyTutorialContent」の**「getURI(String)」**ノードを開きます。

   1 つのパラメータ（path）と、1 つのメソッド戻り属性 URI がこの場所にリストされます（図 7-6）。

図 7-6 データ・コントロール・パレット - MyTutorialContent.getURI



3. **「URI」**ノードを選択し、MyContent.jspx にドラッグ・アンド・ドロップします。

   データ・コントロール・パレットから項目をドラッグしてページにドロップすると、JDeveloper に適切なコンポーネントのコンテキスト・メニューが表示されます。

4. コンテキスト・メニューから**「リンク」**を選択し、**「ADF 実行リンク」**を選択します（図 7-7 を参照）。

図 7-7 getURI の JDeveloper コンテキスト・メニュー

5. C:¥TutorialContent¥help.html にある「ヘルプ」ページへのリンクを公開するには、/help.html を入力することにより、ファイルを指し示す **path** パラメータを使用します。

   必ずスラッシュを含めてください（図 7-8）。

図 7-8 GetURI - アクション・バインディング・エディタ

6. **「OK」**をクリックします。

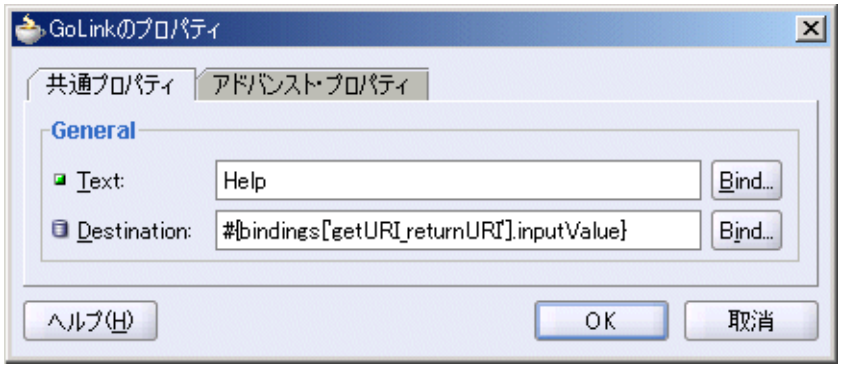
   MyContent.jspx 上に、デフォルトのリンク・テキスト「goLink1」を持つ新しい goLink が表示されます。

7. 構造ウィンドウで、**「af:goLink - goLink1」**をダブルクリックして、デフォルト・プロパティを編集します。

8. デフォルトのリンク・テキストは「goLink1」です。このテキストを Help という語で置き換えます（図 7-9）。

   「宛先」フィールドの式（#{bindings['getURI_returnURI'].inputValue}）は、「ヘルプ」ページの URI をフェッチします。

図 7-9 GoLink プロパティ - ヘルプ・リンクの構成



9. **「OK」**をクリックして、GoLink のプロパティ・ウィンドウを閉じます。

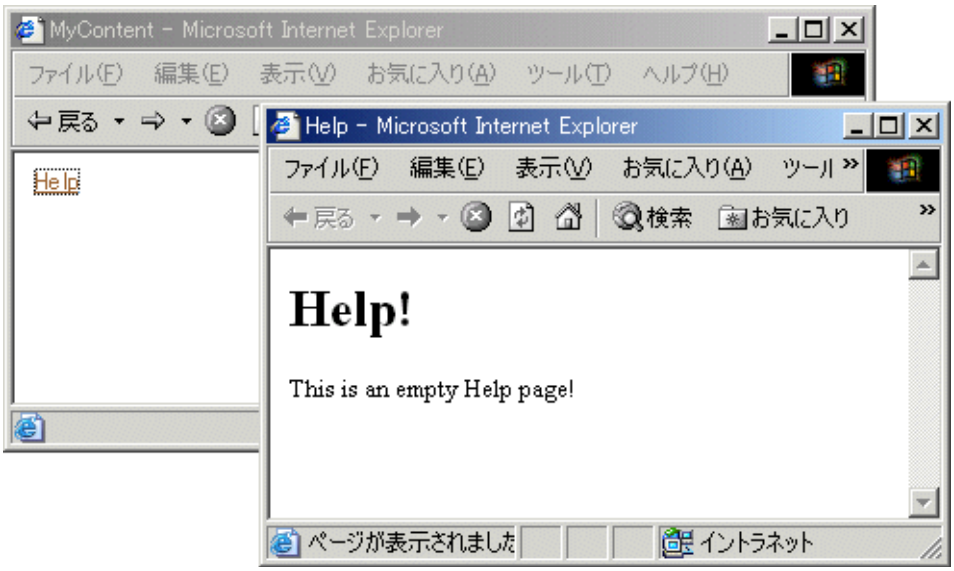
10. JDeveloper のツールバーで、**「すべて保存」**アイコンをクリックします。

11. ページを実行する前に、埋込み OC4J サーバーを停止します。メイン・メニューから**「実行」**、**「終了」**、**「埋込み OC4J サーバー」**の順に選択します。

12. 「**MyContent.jspx**」を右クリックし、「**実行**」を選択して、「ヘルプ」リンクが機能することを確認します。

    ブラウザ・ウィンドウにページが表示されるとき、新しい「ヘルプ」リンクが表示されます。リンクをクリックして、正しいファイルが表示されることを確認します。ブラウザは、図 7-10 のように表示されます。

図 7-10 ファイル・システムのデータ・コントロールを介したファイル・リンク



手順 3: 表でのフォルダ・コンテンツの表示

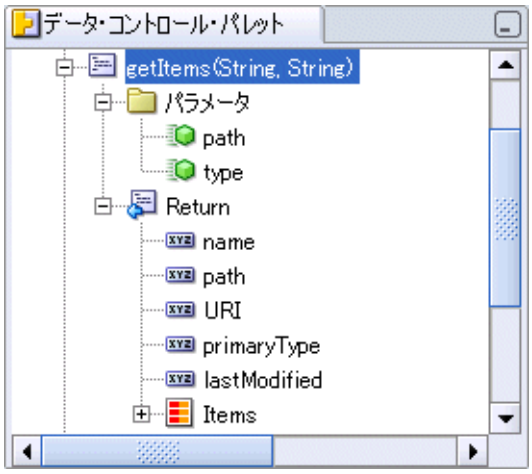
この手順では、データ・コントロール・メソッド `getItems` を使用して、表でファイルおよびフォルダの情報を公開する方法を学びます。次に、`C:¥TutorialContent` に存在するすべてのファイルがリストされた表（図 7-11 に示すような表）を作成します。表内の各ファイル名には、実際のファイルへのハイパーリンクを含めます。

図 7-11 ハイパーテキスト・リンクとして表示されたファイル・システム・コンテンツ

1. アプリケーション・ナビゲータで、「**MyContent.jspx**」をダブルクリックして、ビジュアル・エディタでページを開きます。

2. データ・コントロール・パレットで、「**MyTutorialContent**」の「**getItems**」ノードを開きます。2 つのパラメータ（`path` および `type`）と「`Return`」オプションが表示されます（図 7-12 を参照）。

図 7-12 データ・コントロール・パレット - MyTutorialContent.getItems

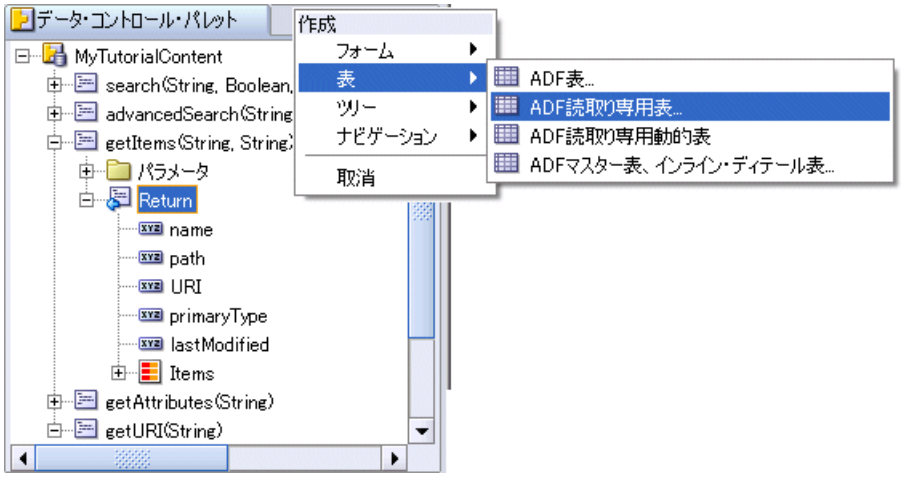


3. まずは、このデータ・コントロールを介して使用可能なすべてのファイルとフォルダがリストされた表を作成します。これを行うには、「`Return`」ノードを選択してページにドラッグし、「ヘルプ」リンク（「**af:goLink - Help**」）の下にドロップします。

   データ・コントロール・パレットから項目をドラッグしてページにドロップすると、JDeveloper に適切な UI コンポーネントのコンテキスト・メニューが表示されます。ファイル・システムの項目は、フォーム、表、ツリーまたはナビゲーション項目として表示できます。

4. コンテキスト・メニューから「**表**」を選択し、「**ADF 読取り専用表**」を選択します（図 7-13 を参照）。

図 7-13 getItems の JDeveloper コンテキスト・メニュー

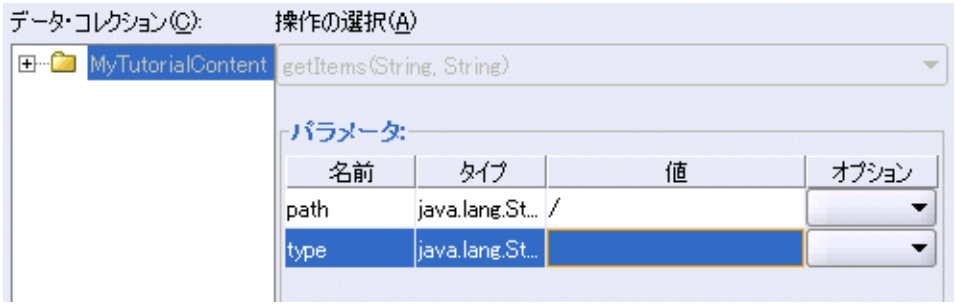


   アクション・バインディング・エディタが表示されます（図 7-14）。

5. データ・コントロールのベース・パスの下のすべての内容を表示するには、`path` パラメータに `/` を入力します。必ず、ここには円記号ではなくスラッシュを入力してください。

6. この時点では「タイプ」は空白のままにしておいてください。これによって、表にファイルとフォルダの両方を表示することを指定します。後で、ファイルのみを表示するように表を構成します。

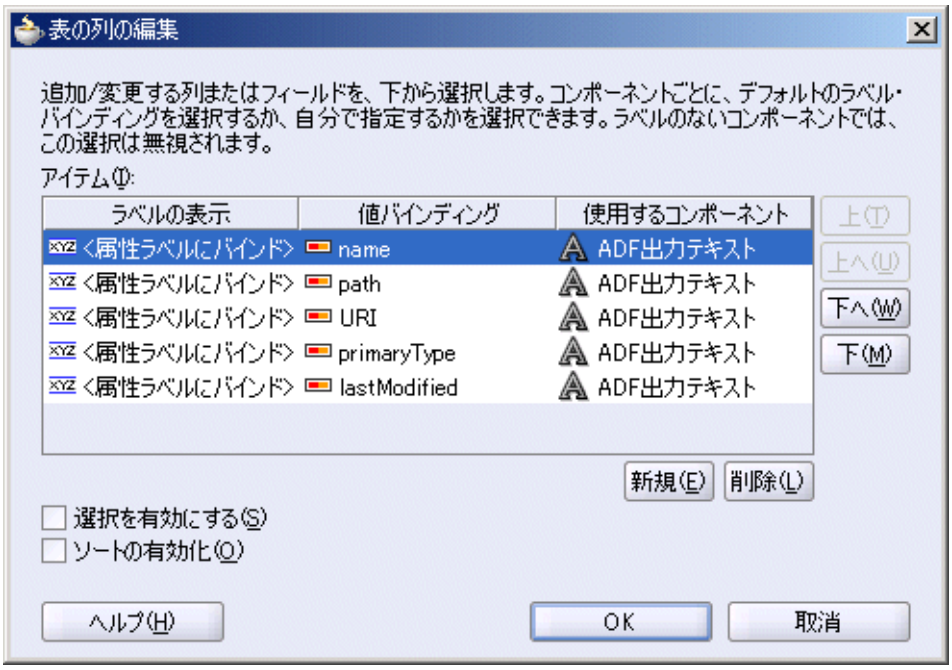
図 7-14 getItems - アクション・バインディング・エディタ



7. 「**OK**」をクリックします。

   表の列の編集ウィンドウが表示されます（図 7-15）。

図 7-15 表の列の編集 - デフォルト



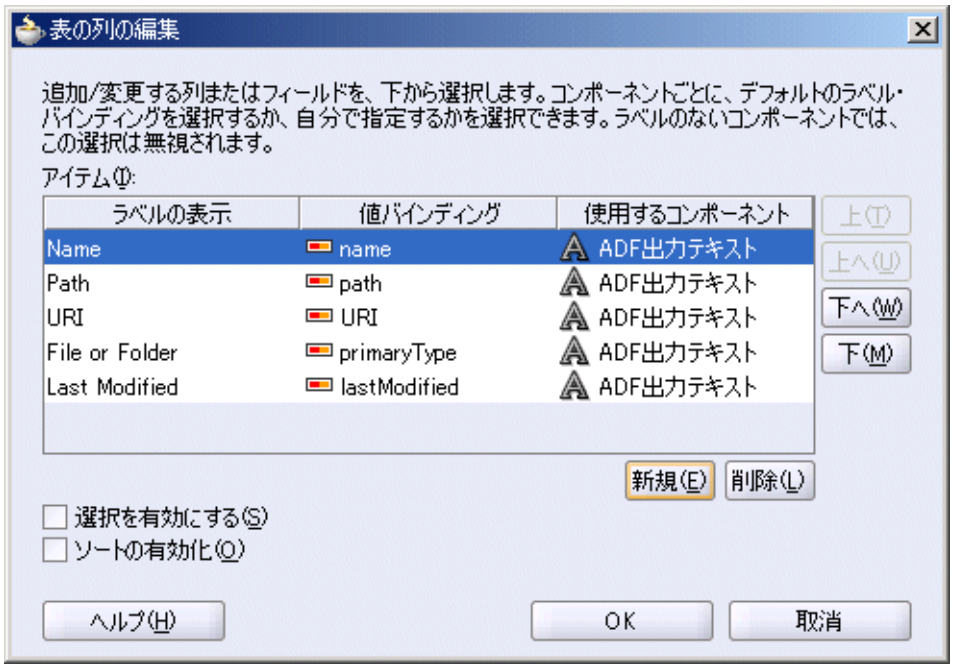
ここで、このページで使用可能な表示オプションをいくつか示します。ファイルおよびフォルダの名前（`name`）に加えて、`path`、`URI`、`primaryType`、`lastModified` など、他のコンテンツ属性も公開できます。デフォルトでは、すべての属性がデフォルト・ラベルとともに公開されますが、表示されるコンテンツはカスタマイズできます。詳細なニーズに応じて、属性の削除、表示ラベルの編集、および表示順序の変更を行うことができます。

ここでは、「ラベルの表示」を、わかりやすいラベルに編集するのみにします。

8. （「**name**」の隣の）「**< 属性ラベルにバインド >**」をクリックし、`Name` を入力します。

   次に、他の属性（path、URI、primaryType、lastModified）の表示ラベルを編集します。`Path`、`URI`、`File or Folder` および `Last Modified` のように、新しい表示ラベルを入力します（図 7-16）。

図 7-16 表の列の編集 - 表示ラベルのカスタマイズ



9. 「**OK**」をクリックします。

   MyContent.jspx の表が、図 7-17 のように表示されます。

図 7-17 フォルダ・コンテンツを公開するための読取り専用表

Name	Path	URI	File or Folder	Last Modified
#{row.name}	#{row.path}	#{row.URI}	#{row.primaryType}	#{row.lastModified}
#{row.name}	#{row.path}	#{row.URI}	#{row.primaryType}	#{row.lastModified}
#{row.name}	#{row.path}	#{row.URI}	#{row.primaryType}	#{row.lastModified}

10. 今度は、ブラウザでページを表示します。「**MyContent.jspx**」を右クリックし、「**実行**」を選択します。

    ページがブラウザ・ウィンドウに表示されると、MyTutorialContent データ・コントロールを介して使用可能な（たとえば、ディレクトリ `C:¥TutorialContent` の下にある）すべてのファイルおよびフォルダのリストが表示されます（図 7-18 を参照）。

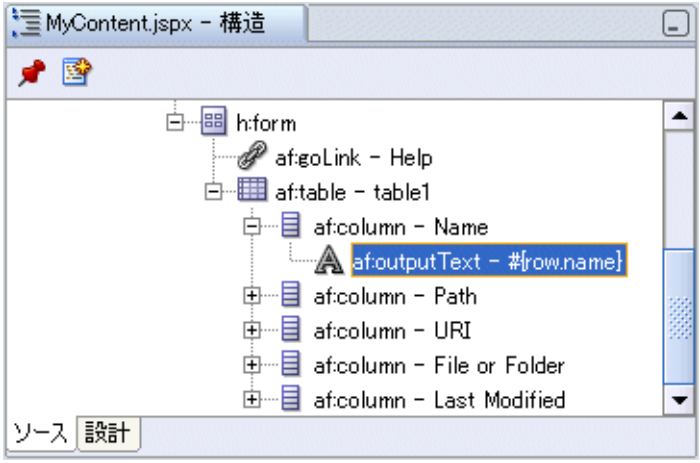
図 7-18 ブラウザ - 表に表示されたフォルダ・コンテンツ

Name	Path	URI	File or Folder	Last Modified
More_Content	/More_Content	/get/conn/MyTutorialContent/path/More_Content	nt:folder	
More_Images	/More_Images	/get/conn/MyTutorialContent/path/More_Images	nt:folder	
help.html	/help.html	/get/conn/MyTutorialContent/path/help.html	nt:file	2006/07/25
welcome.html	/welcome.html	/get/conn/MyTutorialContent/path/welcome.html	nt:file	2006/10/18
hula.gif	/hula.gif	/get/conn/MyTutorialContent/path/hula.gif	nt:file	2006/11/27
camera.gif	/camera.gif	/get/conn/MyTutorialContent/path/camera.gif	nt:file	2006/11/27

デフォルトでは、表にファイルおよびフォルダの属性が読取り専用テキスト（`af:outputText`）として表示されます。次の手順では、「名前」属性を ADF GoLink（`af:goLink`）として表示する方法を学びます。

11. 構造ウィンドウ（図 7-19 を参照）で、表の最初の列（「`af:column - Name`」）を開くと、デフォルトの表示形式（「`af:outputText - #{row.name}`」）が表示されます。

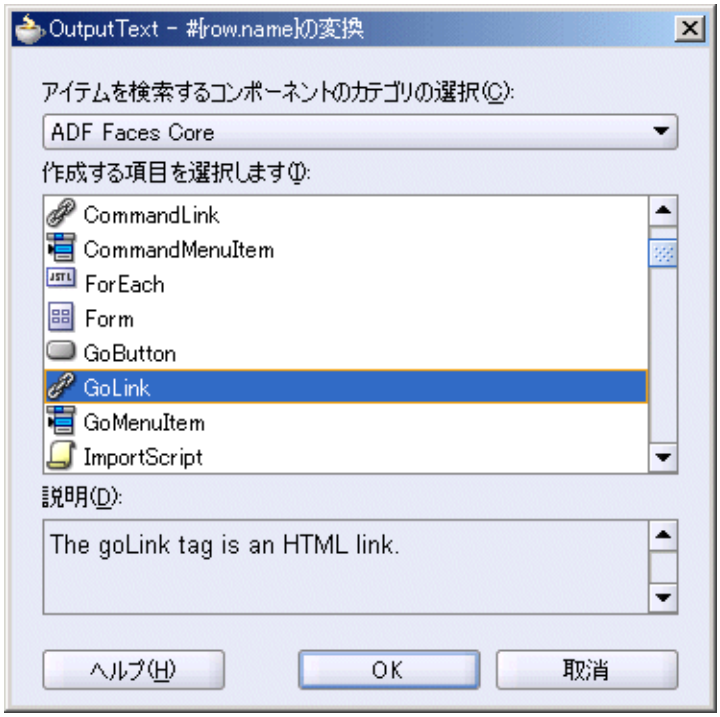
図 7-19「名前」列のデフォルト表示形式



12. 「**af:outputText - #{row.name}**」を右クリックし、「**変換**」をクリックします。
13. プルダウン・メニューから、「**ADF Faces Core**」を選択します。

14. 「**GoLink**」を選択します（図 7-20）。

図 7-20 OutputText を GoLink に変換



15. 「**OK**」をクリックし、再度「**OK**」をクリックして変換を確認します。

    このチュートリアルでは、ファイルおよびフォルダの名前（name）がハイパーリンクとして表示されるように GoLink を設定することにします。これを行うには、デフォルトの GoLink プロパティを編集する必要があります。

16. 構造ウィンドウで、新しい「**af:goLink - goLink 1**」を右クリックし、「**プロパティ**」をクリックしてデフォルト設定を表示します。

17. デフォルトのリンク・テキストは、「goLink 1」です。デフォルトのかわりにファイルおよびフォルダの名前（name）を表示するには、バインディング・エディタを使用して、必要な式（`#{row.name}`）を構築します（図 7-21 を参照）。

    a. 「**テキスト**」フィールドで、「**バインド**」をクリックします。

    b. 「**JSP オブジェクト**」を開き、「**行**」を開きます。

    c. 「**名前**」をダブルクリックして、式「`#{row.name}`」を選択します。

    d. 「**OK**」をクリックします。

18. HTTP ハイパーリンクの URI を生成するには、次のようにします。

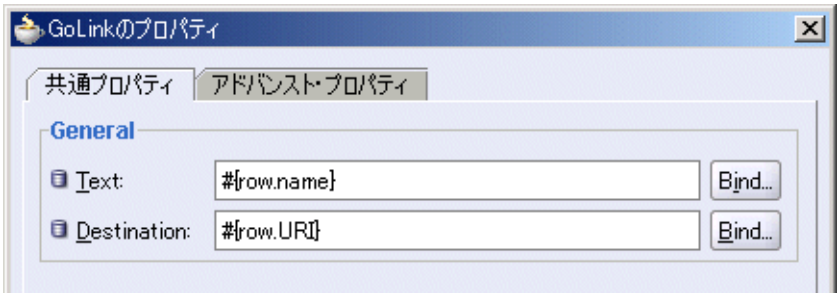
    a. 「**宛先**」フィールドで、「**バインド**」をクリックします。

    b. 「**JSP オブジェクト**」を開き、「**行**」を開きます。

    c. 「**URI**」をダブルクリックして、式「`#{row.URI}`」を選択します。

    d. 「**OK**」をクリックします。

図 7-21 GoLink プロパティ - ファイル名をハイパーリンクとして構成



19. 再度ページを実行する前に、埋込み OC4J サーバーを停止します。メイン・メニューから**「実行」**、**「終了」**、**「埋込み OC4J サーバー」**の順に選択します。
20. 再度 **MyContent.jspx** を実行します。

    今度は、図 7-22 に示すようなハイパーリンク付きのファイルおよびフォルダ名のリストが表示されます。

図 7-22 ブラウザ - ハイパーリンクとして表示されたフォルダ・コンテンツ

Name	Path	URI	File or Folder	Last Modified
More Content	/More_Content	/get/conn/MyTutorialContent/path/More_Content	nt:folder	
More Images	/More_Images	/get/conn/MyTutorialContent/path/More_Images	nt:folder	
help.html	/help.html	/get/conn/MyTutorialContent/path/help.html	nt:file	2006/07/25
welcome.html	/welcome.html	/get/conn/MyTutorialContent/path/welcome.html	nt:file	2006/10/18
hula.gif	/hula.gif	/get/conn/MyTutorialContent/path/hula.gif	nt:file	2006/11/27
camera.gif	/camera.gif	/get/conn/MyTutorialContent/path/camera.gif	nt:file	2006/11/27

21. ファイル名の 1 つをクリックします。

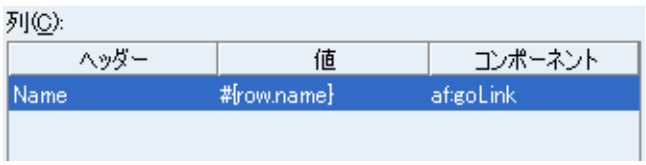
    選択したファイルが、ブラウザ・ウィンドウ内に表示されます。
22. フォルダの名前をクリックします。

    ダイレクト URL を介してフォルダにアクセスできないため、今度は認証エラーが表示されます。フォルダには、データ・コントロールを介してのみアクセスできます。

    次の手順に進む前に、これまでに行った内容をまとめてみます。最初に、MyTutorialContent データ・コントロールに基づいて表を作成しました。デフォルトで、表にファイル・システム情報が書式設定されていないプレーン・テキストで公開されることがわかりました。次に、ファイルおよびフォルダの名前に書式設定を適用して、これらをハイパーテキスト・リンクとして表示しました。

    最後に、表を整理してみます。「名前」列のみが表示されるように表を構成し、表示をファイルに限定します。
23. JDeveloper に戻って、「名前」列のみが表示されるように表を構成します。
    a. 構造ウィンドウで、**「af:table -table1」**ノードを右クリックし、**「プロパティ」**を選択します。
    b. **「列のサマリー」**タブをクリックします。**「削除」**ボタンを使用して、「Name」列以外をすべて削除します（図 7-23）。

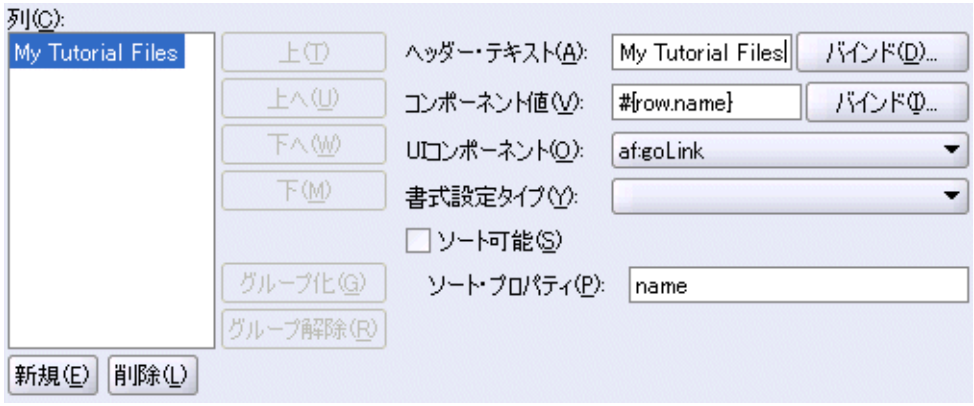
図 7-23 表のプロパティ - 列の編集



c. 「**列の詳細**」タブをクリックします。

d. 「ヘッダー・テキスト」に `My Tutorial Files` を入力します（図 7-24 を参照）。

図 7-24 表のプロパティ - 列表示オプションの編集



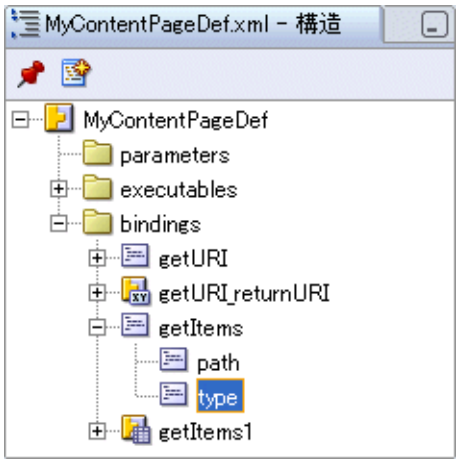
e. 「**OK**」をクリックします。

24. （フォルダでなく）ファイルのみが表示されるように表を構成するには、ページ定義ファイルを編集する必要があります。

a. 「**MyContent.jspx**」を右クリックし、「**ページ定義に移動**」を選択します。

b. 構造ウィンドウで、「**bindings**」および「**getItems**」を開きます（図 7-25）。

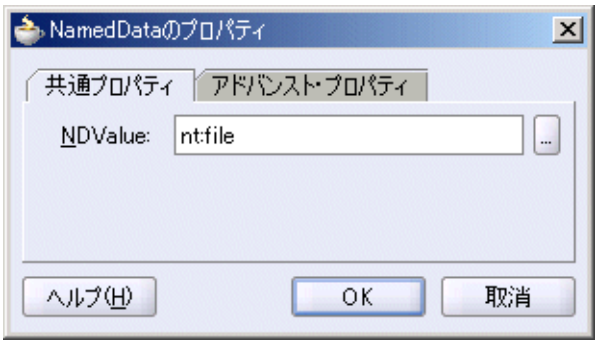
図 7-25 MyContentPageDef.xml での type プロパティの構成



c. 「**type**」をダブルクリックします。

d. 「type」オプションは、`nt:file` と `nt:folder` です。ファイルのみが表示されるように指定するには、**「NDValue」**フィールドに `nt:file` を入力し、**「OK」**をクリックします（図 7-26 を参照）。

図 7-26 NDValue - ファイル表示のみの構成



25. ここで、ページ定義ファイル内にデフォルトで設定されているデータ・コントロールのバインディング設定をいくつか調べます。この設定は独自のアプリケーションに使用する可能性があるので、この設定による結果を理解しておくことが重要です。

    a. 構造ウィンドウで、**「executables」**セクションを開きます。

    b. メソッド・イテレータ**「getItemsIter」**を選択します。

    c. **「表示」**メニューから、**「プロパティ・インスペクタ」**を選択して、デフォルト設定を表示します。

    d. **RangeSize** プロパティは、各ページに表示されるファイルおよびフォルダ項目の数を制御します。このチュートリアルでは、デフォルトの RangeSize の 10 を保持します。

    e. **CacheResults** プロパティは、テーブル・コンテンツがキャッシュされるかどうかを決定します。デフォルト（`true`）では、結果はキャッシュされますが、コンテンツが頻繁に変更されるためにリアルタイムな更新が重要であるようなアプリケーションの場合、この動作では不都合なこともあります。このプロパティを `false` に設定してください。

       ページを実行すると、ページがリフレッシュされるたびにファイル・リストが動的に更新されることがわかります。

26. JDeveloper のツールバーで、**「すべて保存」**アイコンをクリックします。

27. 再度ページを実行します。

    ファイルのリストが、図 7-27 のように表示されます。今度は、フォルダは表示されません。

図 7-27 ブラウザ - ハイパーテキスト・リンクとして表示されたファイルのみ

28. キャッシングが無効になっていることをテストします。ファイル・システム上のファイルの 1 つを名前変更し、ブラウザをリフレッシュします。

    テーブル・コンテンツをキャッシュしないように選択したため、新しいファイル名が即時に表示されます。

このレッスンでは、表にファイル・システム・コンテンツを公開する方法を学びました。次のレッスンでは、同じコンテンツをツリー内に表示します。

手順 4: ツリーでのフォルダ・コンテンツの表示

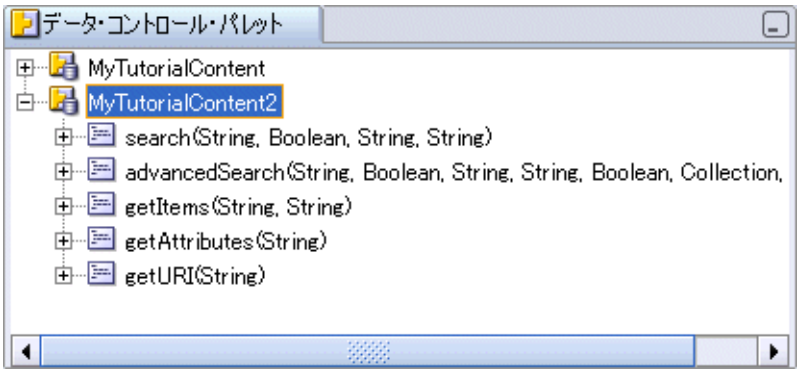
この手順では、`getItems` メソッドを使用して、ファイルおよびフォルダのコンテンツを階層ツリー形式で公開します。C:¥TutorialContent にあるファイルを表示するツリー（図 7-28 に示すようなツリー）を作成し、ツリー内の各ファイル名には、実際のファイルへのハイパーリンクを設定します。しかしまずは、新しいデータ・コントロールを作成します。

図 7-28 ツリーに公開されたファイル・システム・コンテンツ

1. アプリケーション・ナビゲータで、**「Model」**を右クリックし、**「新規」**を選択します。
2. **「Business Tier」**ノードを開き、**「コンテンツ・リポジトリ」**を選択します。
3. **「コンテンツ・リポジトリのデータ・コントロール」**を選択し、**「OK」**をクリックします。
4. **「次へ」**をクリックして、「ようこそ」ページをスキップします。
5. データ・コントロールに名前を付けます。`MyTutorialContent2` と入力し、**「次へ」**をクリックします。
6. このデータ・コントロールを使用してファイル・システムからコンテンツを公開するので、**「リポジトリ・タイプ」**ドロップダウン・リストをクリックし、**「ファイル・システム」**を選択します。
7. 「ベース・パス」フィールドで、先に解凍したコンテンツのパス（たとえば、`C:¥TutorialContent` など）を入力し、**[Enter]** を押します。
8. **「テスト」**ボタンをクリックして、接続詳細を適切に入力したかどうかを確認します。「成功しました。」というメッセージが表示されます。
9. エラー・メッセージが表示された場合は、**「OK」**をクリックし、フルパスを指定するように注意して「ベース・パス」を編集します。テストが成功した場合は、**「OK」**をクリックしてメッセージ・ボックスを閉じ、**「終了」**をクリックします。

10. データ・コントロール・パレットで新しいデータ・コントロールが使用可能になりますので、ここでそれを表示します。「表示」メニューから、**「データ・コントロール・パレット」**を選択します（図 7-29）。

図 7-29 データ・コントロール・パレット - MyTutorialContent2

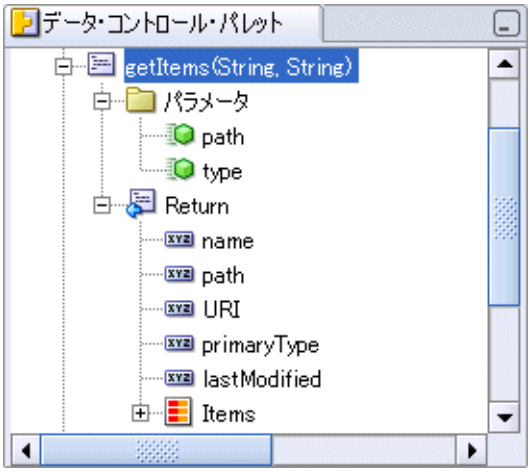


今度は、このデータ・コントロールを介して使用可能なコンテンツを階層ツリー形式で公開します。

11. アプリケーション・ナビゲータで、**「MyContent.jspx」**をダブルクリックして、ビジュアル・エディタでページを開きます。

12. データ・コントロール・パレットで、**「MyTutorialContent2」**の下の**「GetItems」**ノードを開きます（図 7-30）。

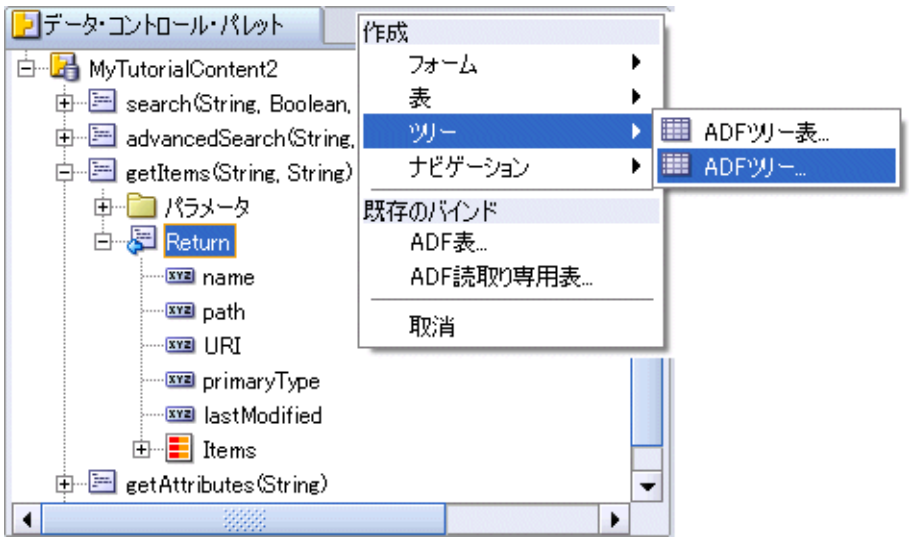
図 7-30 MyTutorialContent2.getItems



13. **「Return」**ノードを選択してページにドラッグし、表（**「af:Table - table1」**）の下にドロップします。

14. コンテキスト・メニューから、「**作成**」、「**ツリー**」、「**ADF ツリー**」の順に選択します（図 7-31 を参照）。

図 7-31 getItems の JDeveloper コンテキスト・メニュー

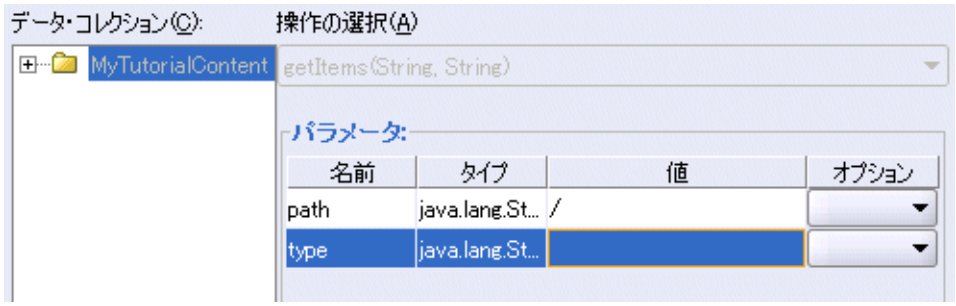


アクション・バインディング・エディタが表示されます。

15. ベース・パス（`C:¥TutorialContent`）の下の内容をすべて表示するツリーを作成するには、`path` パラメータに `/` を入力します（図 7-32 を参照）。必ず、円記号ではなくスラッシュを入力してください。

ツリーにファイルとフォルダの両方を含める必要があるため、「`type`」は空白にしておきます。

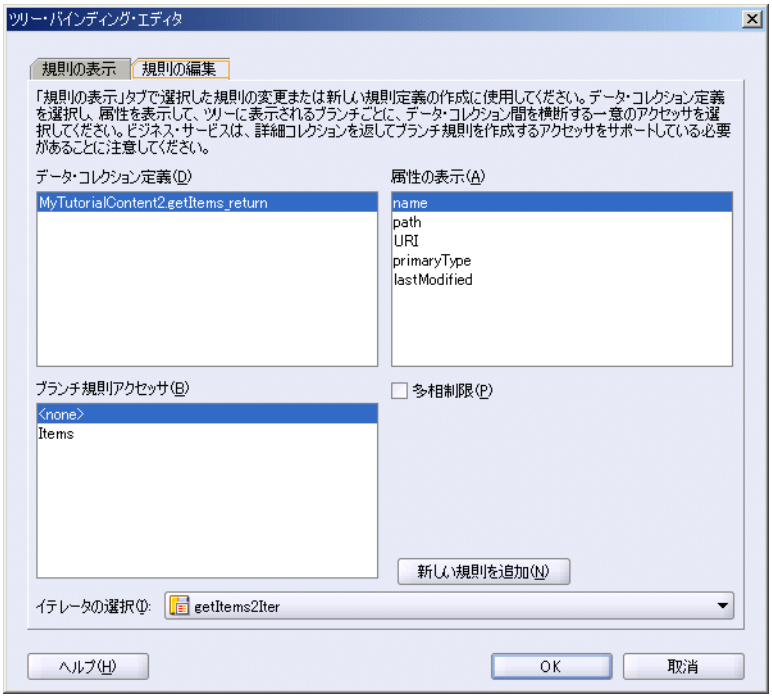
図 7-32 アクション・バインディング・エディタ



16. 「OK」をクリックします。

ツリー・バインディング・エディタが表示されます（図 7-33）。このページについて簡単に説明します。

図 7-33 バインディング・エディタ - デフォルトの「規則の編集」タブ



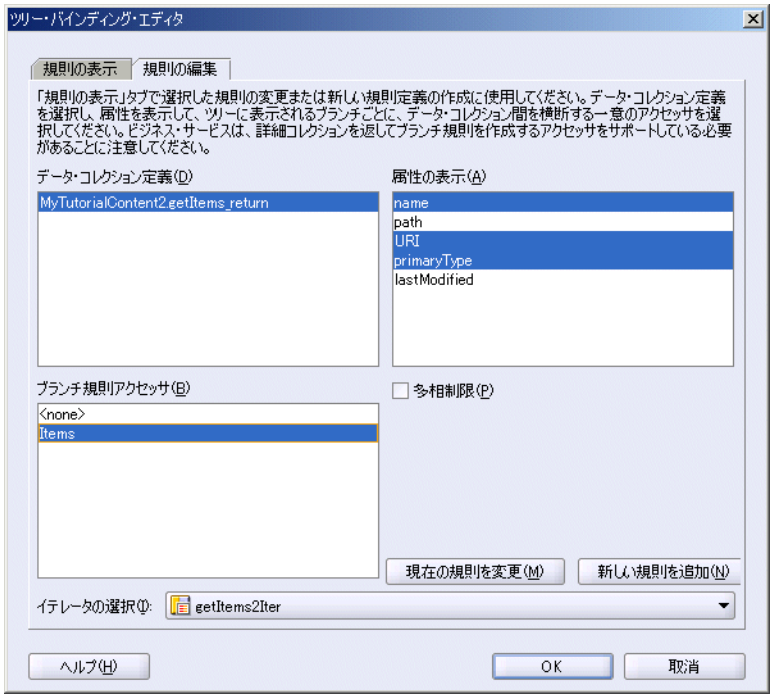
ルールは、ツリー・データ・コントロールが階層データをフェッチおよび表示する方法を定義します。デフォルトのルール設定（図 7-33）では、`getItems` がツリーのルートであること、ツリー・ノード（ブランチ）に `name` 属性のみが表示されること、およびツリー・ノードには子が表示されない（「ブランチ規則アクセッサ」は `<none>`）ことが示されています。

次のいくつかの手順で、これらのデフォルト設定を編集します。また、各ノードの下に階層形式でファイルおよびフォルダを公開してファイル名にハイパーリンクを設定できるようにするルールを定義します。

17. **「属性の表示」**リストで、属性の `name`、`URI` および `primaryType` を複数選択します（図 7-34 を参照）。

    `name` 属性を使用して、ファイルおよびフォルダの名前をツリーに表示します。また、`URI` 属性と `primaryType` 属性を使用して、ファイル・コンテンツへのハイパーリンクを構築します。その他の属性（`path` や `lastModified`）はこのチュートリアルでは使用しないため、ここで選択する必要はありません。

図 7-34 バインディング・エディタ - ルール選択



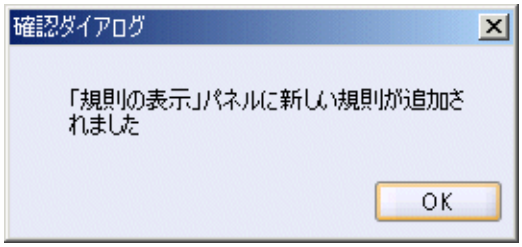
18. 「ブランチ規則アクセッサ」で、**「Items」**を選択します。

    これによって、ツリー・ノードには存在するすべての子が階層形式で表示されます。

19. **「新しい規則を追加」**をクリックします。

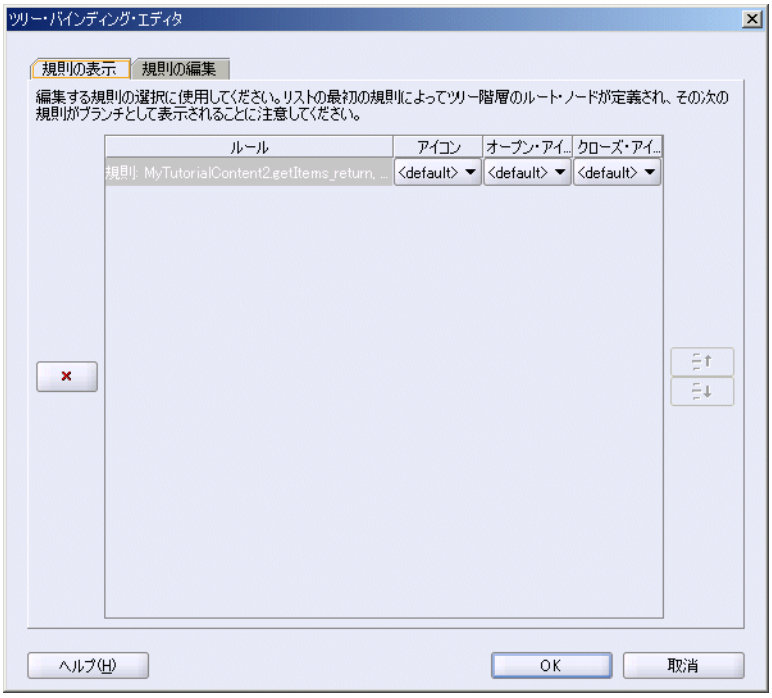
    図 7-35 に示すようなメッセージが表示されます。

図 7-35 新しいルールの確認ダイアログ



20. 「**OK**」をクリックしてメッセージ・ボックスを閉じ、「規則の表示」タブを表示します（図 7-36）。

図 7-36 ツリー・バインディング・エディタ - 「規則の表示」タブ



21. 「**OK**」をクリックします。

これで、MyContent.jspx 上に図 7-37 のようなツリー構造が表示されます。

図 7-37 フォルダ・コンテンツをナビゲートするためのツリー

22. 再度ページを実行する前に、埋込み OC4J サーバーを停止します。メイン・メニューから「**実行**」、「**終了**」、「**埋込み OC4J サーバー**」の順に選択します。

23. 今度は、ブラウザでページを表示します。「**MyContent.jspx**」を右クリックし、「**実行**」を選択します。

    ページがブラウザ・ウィンドウに表示されると、MyTutorialContent2 データ・コントロールを介して使用可能な（たとえばディレクトリ `C:¥TutorialContent` の下にある）ファイルおよびフォルダのリストが表示されます。「`More_Images`」ノードを開き、このサブディレクトリ内のアクセス・コンテンツを確認します（図 7-38）。

図 7-38 ツリーに表示されたフォルダ・コンテンツ

    今度は、`URI` 属性と `primaryType` 属性を非表示にします。この後すぐに、これらの属性を使用してハイパーリンクを構築しますが、これらをツリー内で表示する必要はありません。

24. アプリケーション・ナビゲータで、「**MyContent.jspx**」を選択します。

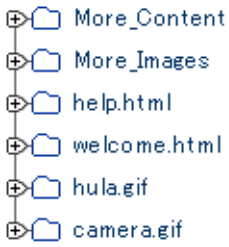
25. 構造ウィンドウで、「**af: tree - tree1**」を右クリックし、「**プロパティ**」を選択します。

26. 「**ノード・スタンプ値**」を `#{node.name}` に変更し、「**OK**」をクリックします。

27. ここで、現在のツリーの外観を確認します。「**MyContent.jspx**」を右クリックし、「**実行**」を選択します。

    ファイルおよびフォルダの名前のみが表示されます（図 7-39）。

図 7-39 ツリーに表示されたファイルおよびフォルダ名

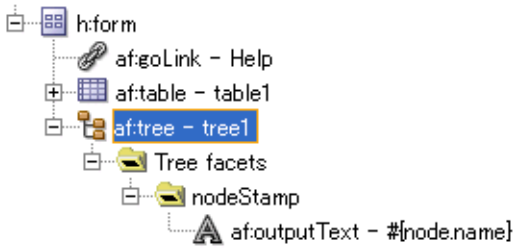


    ツリーには最初の 10 個の項目（デフォルト）が表示されますが、これはページの定義ファイル内の `GetItems` の **RangeSize** プロパティを介してカスタマイズできます。

    デフォルトでは、ツリーにファイルおよびフォルダ名が読取り専用テキストとして表示されますが、表に対して行ったように、これらをハイパーリンクとして表示します。前回と同様、（フォルダでなく）ファイル名にハイパーリンクを追加します。前回とは異なり、フォルダはツリーでのナビゲーションに必要なので、非表示にはできません。そのかわりに、フォルダ名を読取り専用テキストとして表示できます。このデュアル機能に対応するために、（フォルダ用とファイル用に 1 つずつ）2 つのファセットを持つ ADF Faces Switcher コンポーネント（`af:switcher`）を使用します。

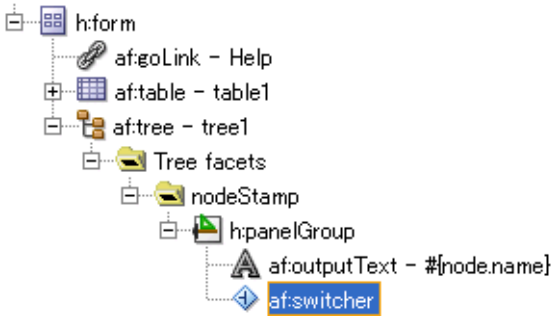
28. 構造ウィンドウで、「**nodeStamp**」ファセットにナビゲートして、現在の表示形式「`af:outputText - #{node.name}`」を表示します（図 7-40）。

図 7-40 ツリーのデフォルト表示形式



29. ADF Switcher コンポーネントを追加します。
    a. 「**nodeStamp**」を右クリックし、「**nodeStamp の中に挿入**」、「**ADF Faces Core**」の順に選択します。
    b. 「**スイッチャ**」を選択し、「**OK**」をクリックします（図 7-41）。

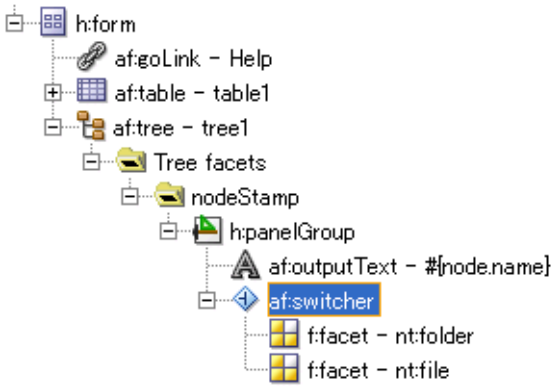
図 7-41 af:switcher コンポーネント



    c. 「**af:switcher**」を右クリックし、「**プロパティ**」をクリックします。
    d. 「FacetName」に `#{node.primaryType}` という式を入力します。
    e. 「**アドバンスト・プロパティ**」タブをクリックします。
    f. 「DefaultFacet」に `nt:file` を入力し、「**OK**」をクリックします。
30. 次に、Switcher の 2 つのファセットを挿入します。
    a. 「**af:switcher**」を右クリックし、「**af:switcher の中に挿入**」、「**JSF Core**」、「**ファセット**」の順に選択します。
    b. 最初のファセットに `nt:folder` という名前を付け、「**OK**」をクリックします。

c. 次に、これらの手順を繰り返し、nt:file という名前の 2 つ目のファセットを追加します（図 7-42）。

図 7-42 2 つのファセットを持つ Switcher コンポーネント

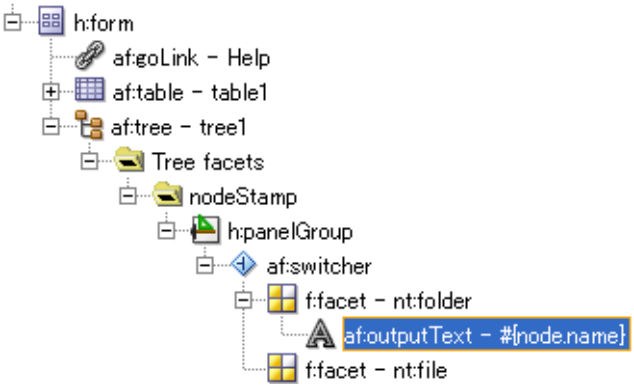


31. フォルダ名に追加の書式設定は必要ありません。デフォルトの表示形式 af:outputText -#{node.name} を再利用して、フォルダ名をプレーン・テキストで表示します。

a. 構造ウィンドウで、「**af:outputText - #{node.name}**」を選択します。

b. 「af:outputText」コンポーネントを「**f:facet - nt:folder**」の上にドラッグ・アンド・ドロップします（図 7-43）。

図 7-43 nt:folder ファセットを outputText として表示



32. af:goLink コンポーネントを使用して、ファイル名をハイパーリンクとして表示します。

a. 「**f:facet - nt:file**」を右クリックし、「**f:facet - nt:file の中に挿入**」、「**ADF Faces Core**」の順に選択します。

b. 「**GoLink**」を選択し、「**OK**」をクリックします。

c. 「**af:goLink - goLink 1**」を右クリックし、「**プロパティ**」をクリックします。

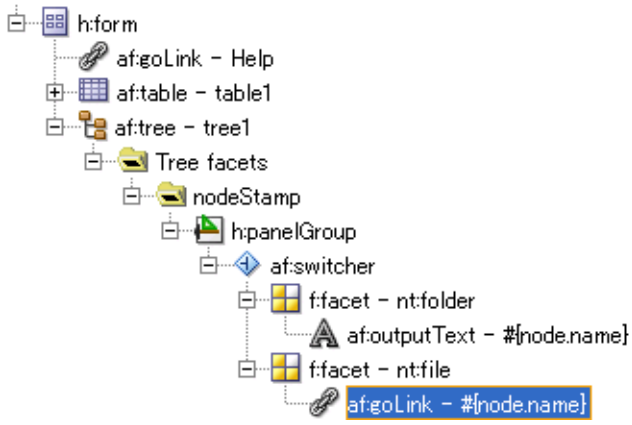
d. リンクの「テキスト」に、#{node.name} という式を入力します。

e. リンクの「宛先」に、#{node.URI} という式を入力します。

f. 「**OK**」をクリックします。

構造ウィンドウを使用して、`af:switcher` の構成を確認します。図 7-44 のように表示されます。

図 7-44 構成済ファセットを表示する Switcher



33. JDeveloper のツールバーで、**「すべて保存」**アイコンをクリックします。
34. 再度ページを実行します。

    今度は、図 7-45 のようなハイパーリンク付きファイル名のツリーが表示されます。

図 7-45 ツリー内に公開されたコンテンツ

35. 任意のファイル名をクリックします。新しいブラウザ・ウィンドウに、そのファイルのコンテンツが表示されます。

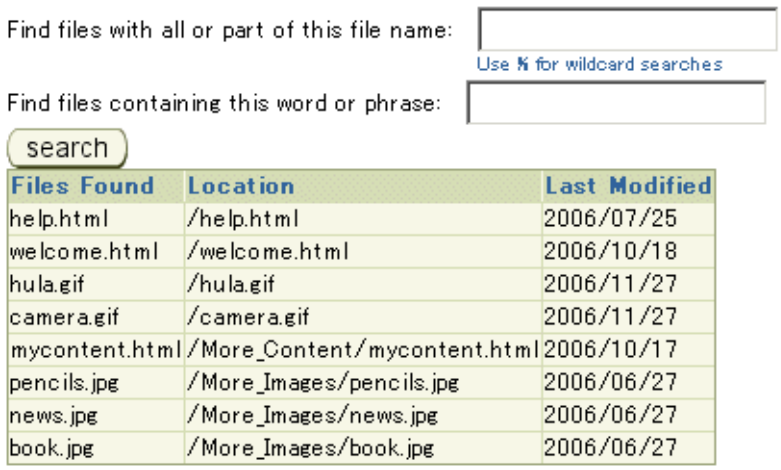
    フォルダ名は、Swithcer コンポーネントの結果としてのプレーン・テキストであることに注意してください。

手順 5: フォルダ・コンテンツの検索

ファイル・システムのデータ・コントロールには、データ・コントロールを介して公開されたデータおよびドキュメントを見つけるための Search メソッドが備わっています。この手順では、このメソッドを使用して、ユーザーがファイル名とキーワードでコンテンツを検索して検索結果を表に表示できるようにする検索フォームを構築します。

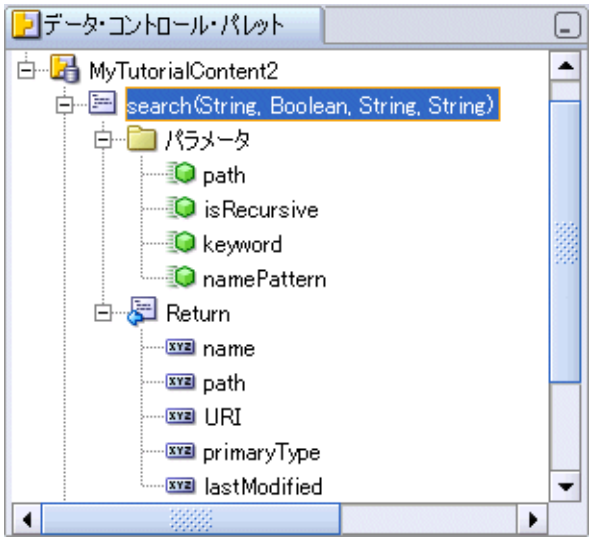
この演習を終えると、検索フォームは図 7-46 のように表示されます。それでは、始めましょう。

図 7-46 データ・コントロール MyTutorialContent2 に基づいた検索フォーム



1. データ・コントロール・パレットで、「**MyTutorialContent2**」の下の「**検索**」ノードを開きます（図 7-47）。

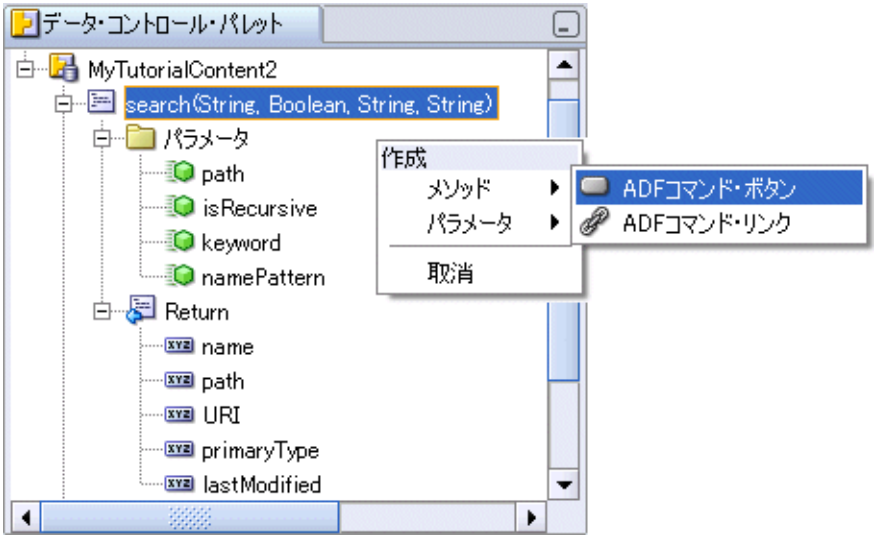
図 7-47 MyTutorialContent2.Search



2. 「**検索**」ノードを選択して構造ウィンドウにドラッグし、ツリー（「**af:Tree - tree1**」）の下にドロップします。

3. コンテキスト・メニューから、「**メソッドの作成**」、「**ADF コマンド・ボタン**」の順に選択します（図 7-48 を参照）。

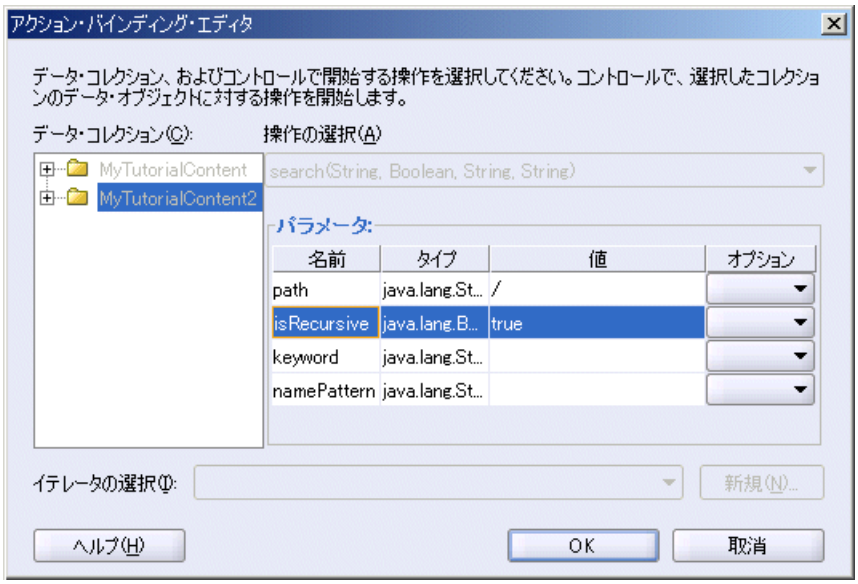
図 7-48 search メソッドの JDeveloper コンテキスト・メニュー



これで、アクション・バインディング・エディタが表示されます。

4. ベース・パス（たとえば、`C:¥TutorialContent` など）の下の内容をすべて検索するため、`path` パラメータに / を入力します。必ず、円記号ではなくスラッシュを入力してください（図 7-49）。

図 7-49 search メソッドのアクション・バインディング・エディタ



5. path 属性を介して指定した位置より下のすべてのサブディレクトリにまで、自動的に検索範囲を拡張するには、「isRecursive」パラメータに `true` を入力します（図 7-49）。

   この検索に対して検索語やファイル名基準を事前定義するわけではないので、「keyword」フィールドと「namePattern」フィールドは空白のままにしておきます。そのかわりに、ユーザーが検索を制御できるように、検索フォーム上にこの両方のパラメータの入力フィールドを設定します。

6. **「OK」**をクリックします。

   「検索」というラベルのコマンド・ボタンがページに表示されます。今度は、このボタンの上に、ユーザーが検索基準を入力できる入力フィールドをいくつか追加します。

7. ユーザーがファイル名で検索できるようにするには、namePattern パラメータを追加します。

   a. データ・コントロール・パレットで、**「パラメータ」**ノードを開き、**「namePattern」**を選択します。

   b. **「namePattern」**パラメータを「検索」ボタンの上にドラッグ・アンド・ドロップします。

   c. 「作成」メニューから、**「テキスト」**、**「ラベル付 ADF 入力テキスト」**の順に選択します。

   d. 構造ウィンドウで、**「af:inputText #{bindings.namePattern...」**をダブルクリックし、適切なラベル（たとえば、`Find files with all or part of this file name:`）を入力します。

   e. 「ヒント」に、`Use % for wildcard searches` を入力します。

   f. **「OK」**をクリックします。

8. ユーザーがファイル・コンテンツ内の句やキーワードを検索できるようにするには、ページに keyword パラメータを追加します。

   a. データ・コントロール・パレットで、**「パラメータ」**ノードを開き、**「keyword」**を選択します。

   b. **「keyword」**パラメータを「検索」ボタンの上にドラッグ・アンド・ドロップします。

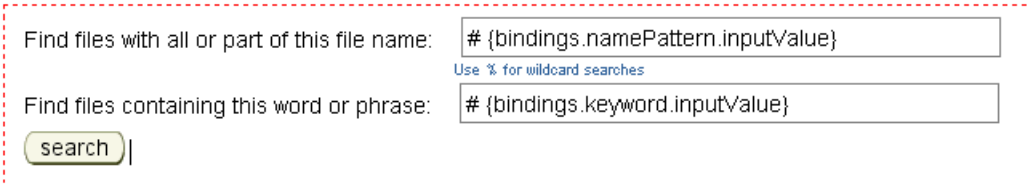
   c. 「作成」メニューから、**「テキスト」**、**「ラベル付 ADF 入力テキスト」**の順に選択します。

   d. 構造ウィンドウで、**「af:inputText #{bindings.keyword...」**をダブルクリックし、適切なラベル（たとえば、`Find files containing this word or phrase:`）を入力します。

   e. **「OK」**をクリックします。

   ページは、図 7-50 のように表示されます。

図 7-50 2 つの入力パラメータを持つ検索フォーム



9. 今度は、検索結果用の表を追加します。

   a. データ・コントロール・パレットで、Search メソッドの下の「**RETURN**」ノードを選択します。

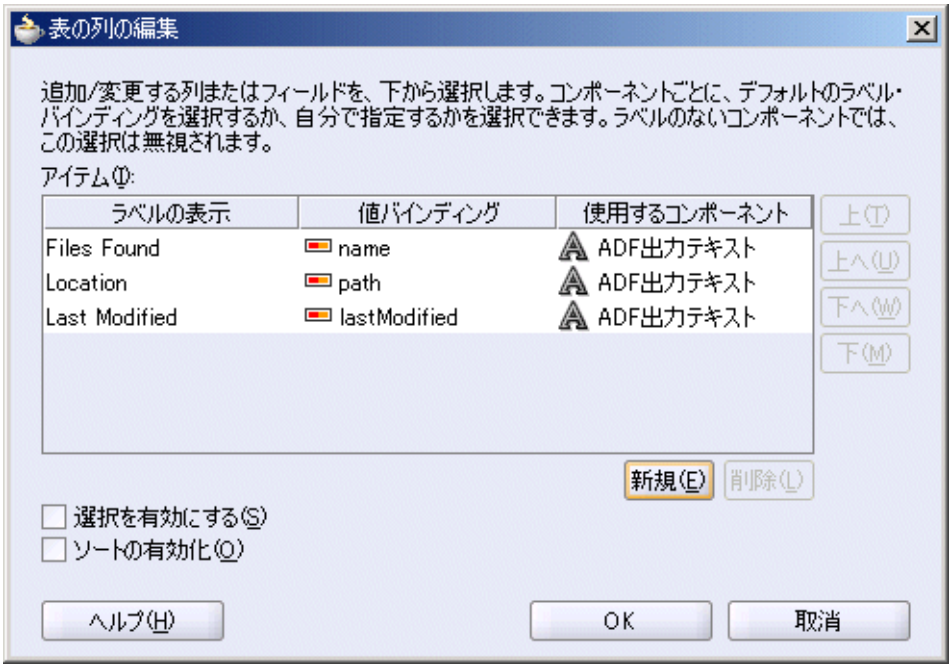
   b. メソッド「**RETURN**」をドラッグして、「検索」ボタンの下にドロップします。

   c. 「作成」メニューから、「**表**」、「**ADF 読取り専用表**」の順に選択します。

   d. 「表の列の編集」ダイアログを使用して、デフォルトの表示設定を編集します。まず、「**URI**」列と「**primaryType**」列を削除します。

   e. 次に、「name」列、「path」列および「lastModifed」列の表示ラベルを編集します。たとえば、`Files Found`、`Location` および `Last Modified` というラベルを入力します（図 7-51 を参照）。

図 7-51 検索結果表の表示ラベルの編集



   f. 「**OK**」をクリックします。

10. JDeveloper のツールバーで、「**すべて保存**」アイコンをクリックします。

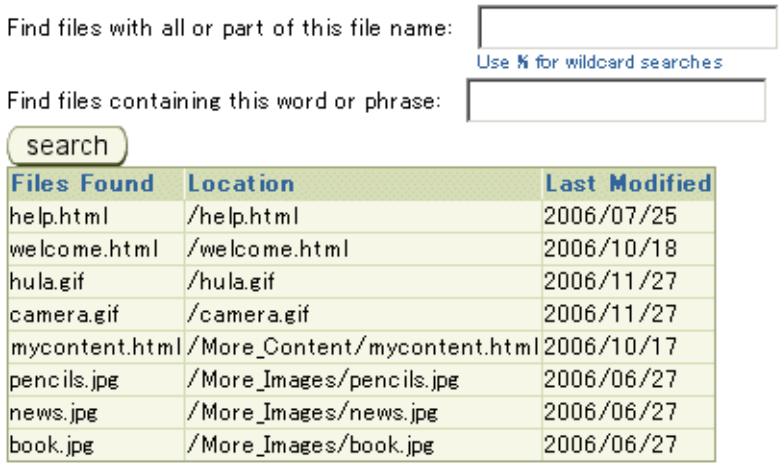
    今度は、ブラウザで検索フォームを表示し、いくつかの検索を実行します。

11. 再度ページを実行する前に、埋込み OC4J サーバーを停止します。メイン・メニューから「**実行**」、「**終了**」、「**埋込み OC4J サーバー**」の順に選択します。

12. 「**MyContent.jspx**」を右クリックし、「**実行**」を選択します。

    図 7-52 に示すような検索フォームが表示されます。

図 7-52 データ・コントロール MyTutorialContent2 に基づいた検索フォーム



13. フォームの最初のフィールドは、`namePattern` パラメータを公開します。ファイル名基準を入力し、「検索」ボタンをクリックし、検索結果表に表示される内容を調べます。指定した名前パターンに一致するファイルのみが表示されます。

    次の検索を試行してください。

    - ファイル拡張子が .html のファイルを検索するには、`%html` を入力します。3 つのファイルがこのパターンに一致します。
    - help という語を含むファイル名を検索するには、`%help%` を入力します。help.html のみが一致します。

14. フォームの 2 つ目のフィールドは、`keyword` パラメータを公開します。このパラメータによって、ファイル内のコンテンツを検索できます。語または句を入力し、「検索」ボタンをクリックし、検索結果表に表示される内容を調べます。指定した語を含むファイルのみが表示されます。

    前述のフィールドから `%help%` を削除し、次のキーワード検索を試行します。

    - WebCenter という語を含むファイルを検索するには、`WebCenter` を入力します。2 つのファイルが表示されます。
    - WebCenter および tutorial という語を含むファイルを検索するには、`WebCenter AND tutorial` を入力します。この両方の語が含まれるのは、1 つのファイルのみです。

これで、コンテンツ統合のレッスンは完了です。WebCenter アプリケーション内にファイル・システム・コンテンツを公開する方法と、検索機能を追加する方法について学びました。この方法の詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

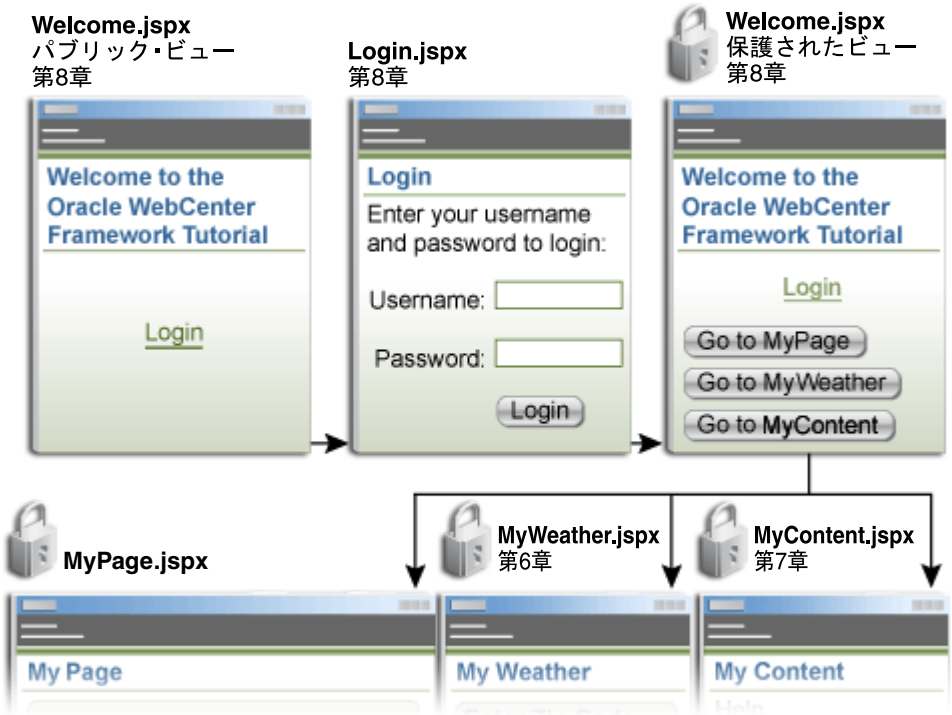
次のレッスンでは、Oracle ADF セキュリティを使用して、このチュートリアルで作成したページを保護します。

8

セキュリティの設定

このレッスンでは、Oracle ADF Security を使用して WebCenter アプリケーションのページを保護する方法を学びます。図 8-1 に、このレッスンを終えた時点でのチュートリアル・アプリケーションの外観を示します。

図 8-1 レッスン 8 を終えた時点でのページ



概要

次の手順で、チュートリアル・アプリケーションにセキュリティを追加し、アプリケーションをテストします。

- 手順 1: ログイン・ページの作成
- 手順 2: ADF Security 設定の構成
- 手順 3: ようこそページの作成
- 手順 4: ページの保護
- 手順 5: orion-web.xml でのセキュリティ・ロールのマッピング
- 手順 6: セキュリティ機能のデモ
- 手順 7: データ・コントロールへのアクセスの認可

前提条件

この章では、埋込み OC4J に付属の軽量 XML リソース・プロバイダ `system-jazn-data.xml` に対してユーザーを認証します。これらの演習を開始する前に、次のユーザー・データをこのファイルに追加する必要があります。

ロール名	ユーザー	説明
page-viewer	Singh	このユーザーは、保護されたページを表示できます。
page-personalizer	Cho	このユーザーは、保護されたページ上のポートレットをパーソナライズできます。
page-customizer	Harvey	このユーザーは、保護されたページをカスタマイズできます。
restricted-user	King	このユーザーは、保護されたページを表示できません。
users	Singh、Cho、King、Harvey、JtaAdmin、oc4jadmin	users ロールは、有効な各ユーザーのリストを保守します。

次のうち 1 つを実行してください。

- サンプルの `system-jazn-data.xml` を、第 2 章の「サンプルの system-jazn-data.xml ファイルのコピー」に示されている JDeveloper の場所にコピーします。サンプルには、この章を完了するために必要なすべてのユーザー・データが含まれています。
- 付録 A「チュートリアルの ID ストアの設定方法」の手順に従って、ユーザー・データを始めから追加します。

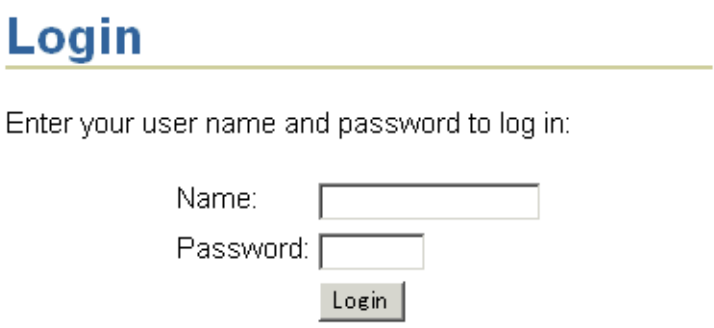
  JDeveloper でセキュアなアプリケーションをすでに構築中で、独自のユーザー・データを `system-jazn-data.xml` に移入した場合は、この方法を使用します。

この準備手順を完了したら、アプリケーションの実際の認証作業に進むことができます。

手順 1: ログイン・ページの作成

この手順では、ユーザー資格証明を受け入れるログイン・ページを作成し、保護されたページへのアクセスを許可します（図 8-2）。このページは、認証されていないユーザーにも表示されます。

図 8-2「Login」ページ



1. アプリケーション・ナビゲータで、**「ViewController」**を右クリックし、**「新規」**を選択します。
2. 「カテゴリ」ペインで、「Web Tier」の下の**「JSP」**をクリックします。

   ここでは、Faces ページのライフサイクルの複雑さを避けるため、ログイン・ページを JavaServer Faces（JSF）ページではなく標準的な JSP として実装することにします。他のアプリケーションでは、アプリケーションにスキニングやポートレットなどの拡張機能が必要になることもあります。Faces ベースのログイン・ページの詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。
3. 「項目」で**「JSP」**をクリックし、**「OK」**をクリックします。
4. **「次へ」**をクリックして、「ようこそ」ページをスキップします。
5. 「ファイル名」フィールドに `Login.jspx` を入力します。
6. 「タイプ」で、**「JSP ドキュメント (*.jspx)」**を選択します。

   これによって、JSP ページ（.jspx）の XML 表示が作成されます。必要に応じて、他のアプリケーション内に JSP ページとしてのログイン・ページを作成することもできます。
7. **「次へ」**をクリックし、再度**「次へ」**をクリックして、エラー・ページ・オプションをスキップします。
8. 「タグ・ライブラリ」ページでは、何も選択しません。必要な場合、「タグ・ライブラリ」ページで二重矢印を使用して、**「選択済のライブラリ」**ペインからライブラリを削除します。
9. **「次へ」**をクリックします。
10. **「終了」**をクリックして、Login.jspx を表示します。
11. **「ソース」**タブをクリックします。

    これから、ログイン・フォームを表示するボディ・タグにコードを追加します。

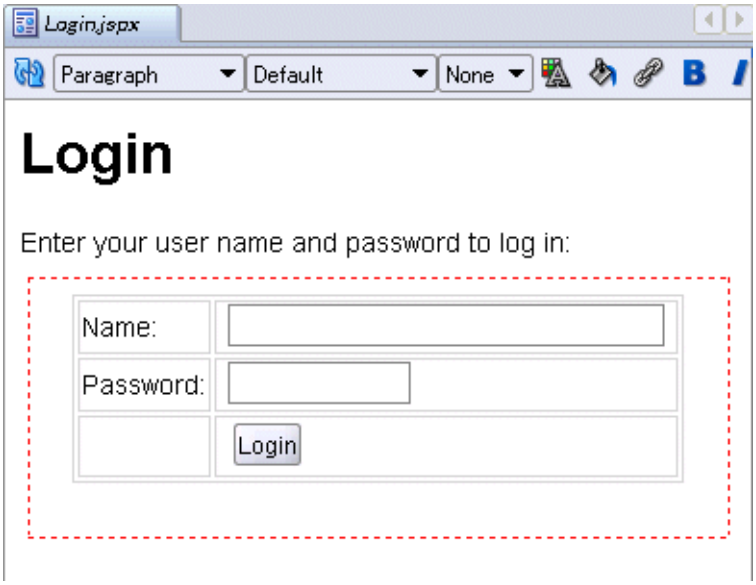
12. 空のボディ・タグ <body></body> を、例 8-1 に示したコードで置き換えます。

例 8-1 ログイン・フォームのコード

```
<body>
     <h1>Login</h1>
     <p>Enter your user name and password to log in:</p>
     <form action='j_security_check' method='post'>
        <table align="center">
          <tr> <td>Name:</td>
            <td> <input type='text' name='j_username'></input> </td>
          </tr>
          <tr>
            <td>Password:</td>
            <td> <input type='password' name='j_password' size='8'></input> </td>
          </tr>
          <tr> <td></td>
            <td> <input type='submit' value='Login'></input> </td>
          </tr>
        </table>
        <br></br>
      </form>
  </body>
```

13. 「設計」タブをクリックして、ログイン・フォームを表示します（図 8-4）。

図 8-3 ログイン・フォームを備えた Login.jspx



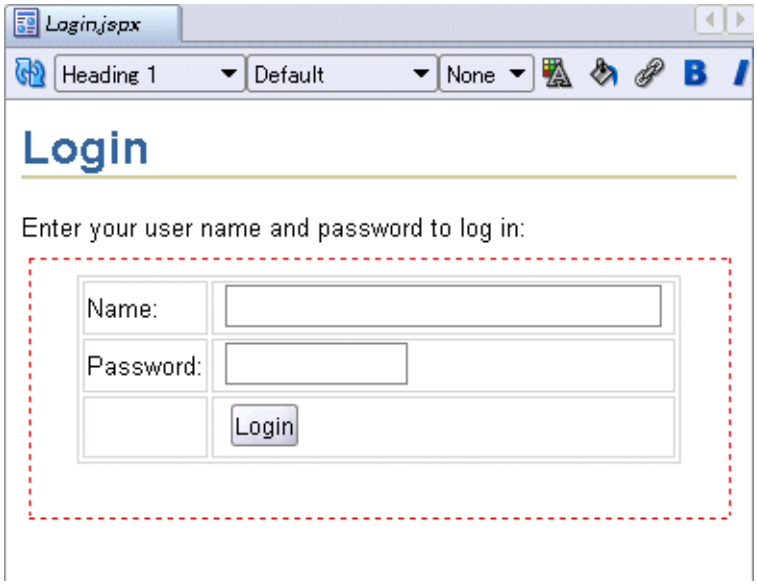
ログイン・ページでは、標準的な J2EE セキュリティ・コンテナ・ログイン・メソッドである j_security_check を使用してユーザー資格証明が検証されます。このセキュリティ確認メソッドは、<form> 要素上にあります。フォームそのものには、2 つの入力フィールドがあります。1 つはユーザー名を受け入れるためのフィールドで、もう 1 つはパスワード用のフィールドです。これらのフィールドに入力された値はそれぞれ、コンテナのログイン Bean 属性 j_username および j_password に割り当てられます。

ログイン・フォーム	名前 / 値
Form Action	j_security_check
Name（テキスト・フィールド）	j_username
Password（テキスト・フィールド）	j_password
Login（送信ボタン）	submit

14. 今度は、スタイルシートをログイン・ページに適用します。コンポーネント・パレットのプルダウン・メニューから、**「CSS」**を選択します。
15. **「JDeveloper」**をドラッグしてページ上にドロップします。

    ログイン・ページはリフレッシュされ、新しいスタイルシートが適用されます（図 8-4 を参照）。

図 8-4 JDeveloper スタイルシートが適用された Login.jspx



次に、Login.jspx を実行して、ブラウザでページを表示します。

16. JDeveloper のツールバーで、**「すべて保存」**アイコンをクリックします。

17. 「**Login.jspx**」を右クリックし、「**実行**」を選択します。

    ブラウザにページが表示されると、ログイン・フォームが表示されます（図 8-5 を参照）。

図 8-5 最終的な Login.jspx

18. ブラウザを閉じ、JDeveloper に戻ります。

今度は、チュートリアル・アプリケーションに Oracle ADF Security オプションを設定します。

手順 2: ADF Security 設定の構成

この手順では、Oracle ADF のセキュリティ・ウィザードを使用して、チュートリアル・アプリケーションに認証設定を構成します。選択したオプションは、`web.xml` または `orion-application.xml` に記録されます。これから行う手順の概要を簡単に示します。

- Oracle ADF の認証を有効化します。
- ユーザー認証用の軽量 XML リソース・プロバイダを選択します。
- 認証用のプロトコルをフォームベースとして指定します。
- adfAuthentication サーブレットへの認証済ユーザー（ValidUsers）アクセス権を付与します。

詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

それでは、ADF セキュリティ・ウィザードを開始します。

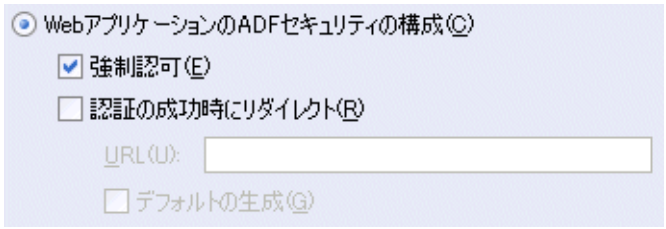
1. アプリケーション・ナビゲータで、「**ViewController**」を選択します。
2. 「**ツール**」メニューから、「**ADF セキュリティ・ウィザード**」を選択します。

   ADF セキュリティ・ウィザードによって、構成プロセスの手順が示されます。
3. 「**次へ**」をクリックして、「ようこそ」ページをスキップします。

4. **「強制認可」**を選択します（図 8-6 を参照）。

   このオプションでは、認証に使用される adfAuthentication サーブレットや、ポリシーの認可実施に必要なその他のサーブレットやフィルタ（リクエストされたページの現行ユーザーの権限のチェックを実施するフィルタ）を構成することによって、ADF Security 機能を構成します。

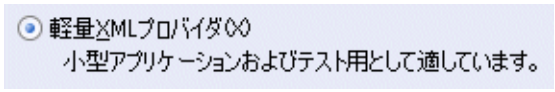
図 8-6 ADF セキュリティ・ウィザード - 認証の有効化



   アプリケーションにログインしているユーザーに対して、認証後に特定のページを表示する場合は、**「認証の成功時にリダイレクト」**を選択し、必要なページを指定します。ただしここでは、このようなチュートリアル・アプリケーションを構成しません。このため、デフォルトの動作（認証済ユーザーを、認証前にアクセスを試行したページに戻す）を使用します。

5. **「次へ」**をクリックして、ウィザードの次のページに進みます。
6. **「軽量 XML プロバイダ」**を選択します（図 8-7）。

図 8-7 ADF セキュリティ・ウィザード - JAAS プロバイダの選択

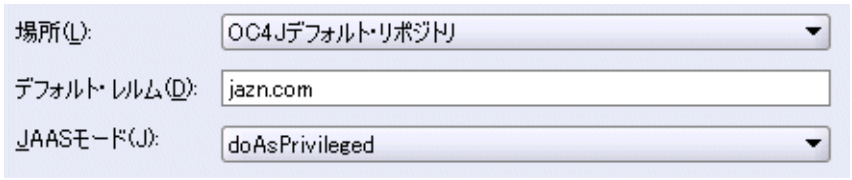


   Oracle ADF Security では、特定のリソース・プロバイダに対してユーザーが認証されます。このチュートリアルでは、レッスンの最初に設定した軽量リソース・プロバイダ `system-jazn-data.xml` を使用します。

7. **「次へ」**をクリックして、ウィザードの次のページを表示します。
8. 「JAAS モード」を**「doAsPrivileged」**に設定します（図 8-8）。

   Oracle ADF Security では、このオプションの設定が必要です。

図 8-8 ADF セキュリティ・ウィザード - XML 設定の構成



   デフォルト・レルムは jazn.com です。このリリースでは、アプリケーション・レベルではなくシステム・レベル（JAZN ファイル）の権限のみが Oracle ADF Security によって読み取られることに注意してください。

9. **「次へ」**をクリックします。

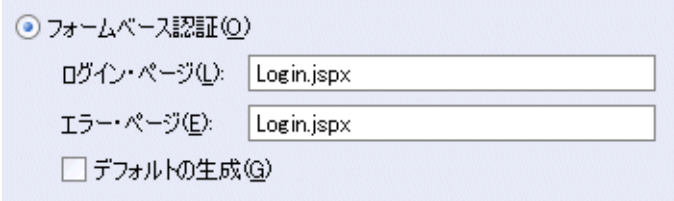
10. 「ログイン」ページで、**「フォームベース認証」**を選択します（図 8-9 を参照）。これによって、チュートリアル・アプリケーションで認証を簡易化するためのフォームを使用することを指定します。

    ログイン・フォームは Login.jspx で使用するため、ログイン・フォームおよびログイン・エラー・メッセージ（`login.html` および `error.html`）用にデフォルトのページを生成する必要はありません。

11. 「ログイン・ページ」に `Login.jspx` を入力します。

12. 「エラー・ページ」にも、`Login.jspx` を入力できます。

図 8-9 ADF セキュリティ・ウィザード - フォームベース認証の構成

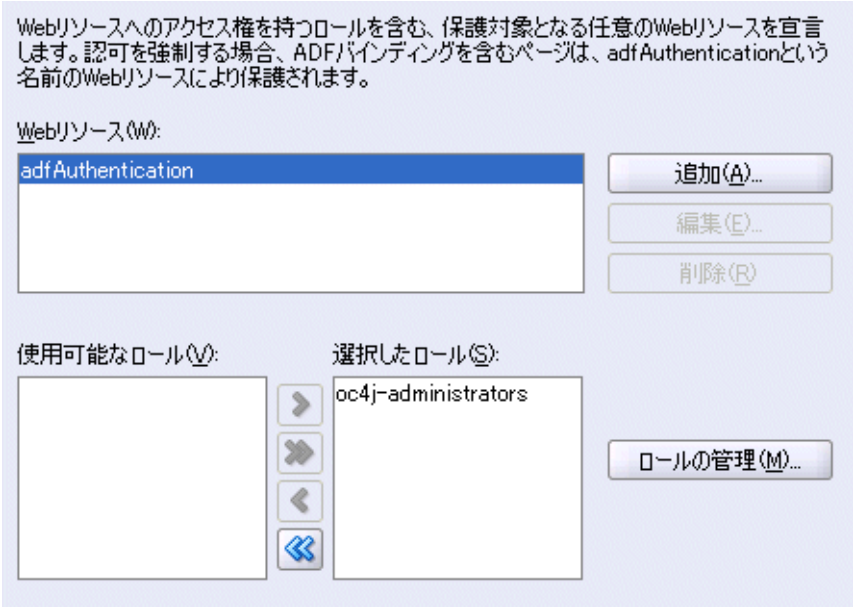


個別にエラー・メッセージを作成することは簡単ですが、このチュートリアルでは同じページを使用します。エラー・ページ（たとえば、LoginError.jspx など）を作成する場合は、Login.jspx と同じページを作成し、認証の失敗を示すエラー・メッセージを追加します。

13. 「次へ」をクリックして、ウィザードの最後のページ（「リソース」）を表示します（図 8-10）。

    このページでは、保護する必要のあるアプリケーション内のリソースを定義し、各リソースにアクセスできる J2EE セキュリティ・ロールを指定します。

図 8-10 ADF セキュリティ・ウィザード - adfAuthentication サーブレットの保護



adfAuthentication リソース（認証サーブレット）が定義されています。このサーブレットは、ログイン URL の既知のエンドポイントとして機能します。このサーブレットそのものは J2EE セキュリティ制約によって保護されているため、ユーザーが現在のアクティブ・セッションなしでこのサーブレットにアクセスしようとすると、ログイン・ページにリダイレクトされます。

このリソースは編集も削除もできませんが、このリソースにアクセスできるロール・セットの指定はできます。

adfAuthentication リソースへの有効なユーザー・アクセスを許可するには、J2EE ロールを作成して `ValidUsers` という名前を付け、このロールにアクセス権を付与する必要があります。これから、この手順を実行します。

14. **「ロールの管理」**をクリックします。
15. **「追加」**をクリックし、`ValidUsers` という名前を入力します（図 8-11）。

    後で、この J2EE ロールを（レッスンの最初に設定したリソース・プロバイダ `system-jazn-data.xml` 内に定義されている）`users` という名前の ID ストア・ロールにマップします。`users` ロールは、有効な各ユーザーのリストを保守します。

    セキュリティの観点から、このロールに権限を割り当てると、認証済のパブリック・リソースを効果的に定義できます。つまり、特定の権限を定義しなくても、すべてのユーザーに対してパブリック・リソースが使用可能になります。

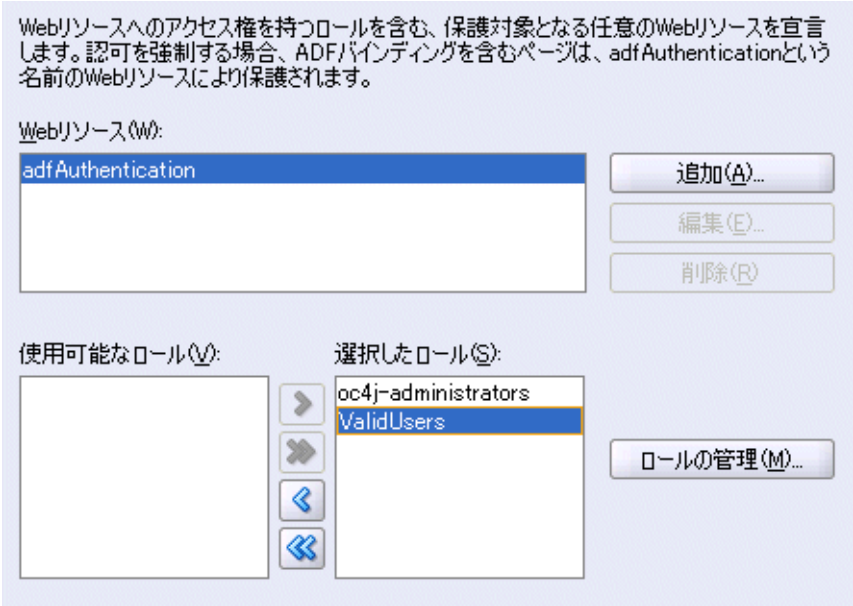
図 8-11 ADF セキュリティ・ウィザード - J2EE セキュリティ・ロールの追加

16. **「OK」**をクリックします。
17. **「閉じる」**をクリックします。

    「使用可能なロール」のリストに、ValidUsers ロールが表示されます。

18. 二重矢印（**「すべて追加」**）をクリックして、**「使用可能なロール」**リスト内のすべてのロールを**「選択したロール」**リストに移動します（図 8-12）。

図 8-12 ADF セキュリティ・ウィザード - adfAuthentication リソースへのアクセス権の付与



これで、ADF セキュリティ・ウィザードの設定は完了です。

19. **「次へ」**をクリックし、**「終了」**をクリックします。

20. JDeveloper のツールバーで、**「すべて保存」**アイコンをクリックします。

このチュートリアルの次の手順に進む前に、`web.xml` および `orion-application.xml` に行った変更内容を確認します。これらのファイルは、アプリケーション・ナビゲータで次の場所に表示されます。

- web.xml は、「ViewController」、「Web コンテンツ」、「WEB-INF」の下にあります。
- orion-application.xml は、「ViewController」、「アプリケーション・ソース」、「META-INF」の下にあります。

ファイル名をダブルクリックして、XML エディタでファイルを表示します。

> **ヒント：** タブをダブルクリックすると、そのタブを JDeveloper ウィンドウいっぱいに表示できます。再度ダブルクリックすると、この機能がオフに切り替わります。

ファイルの変更内容を確認するには、履歴ツールを使用する方法があります。「履歴」タブをクリックすると、追加した内容が右側に強調表示されます。強調表示された箇所をクリックして個別に変更することも、**「次の差分に移動」**アイコンを使用して、追加した内容を 1 つずつスクロールすることもできます。これらのエントリの詳細は、『Oracle WebCenter Framework 開発者ガイド』のセキュリティに関する章を参照してください。

今度は、チュートリアル・アプリケーションのようこそページを作成します。

手順 3: ようこそページの作成

この手順では、チュートリアル・アプリケーションの開始ページとなる、ようこそページを作成します。認証されていないユーザーがようこそページを表示すると、ユーザーを認証用のログイン・ページに送る簡単なログイン・リンクが表示されます。これは、ページのパブリック・ビューです（図 8-13）。

図 8-13 ようこそページ - パブリック・ビュー

認証済のユーザーには、ようこそページに別の情報が表示されます。ログイン後、認証済ユーザーはようこそページに再度リダイレクトされますが、今度は保護されたページへのリンクとログアウト・リンクが表示されます（図 8-14 を参照）。

図 8-14 ようこそページ - 保護されたビュー

それでは、ようこそページを作成します。

1. アプリケーション・ナビゲータで、**「ViewController」**を右クリックし、**「新規」**を選択します。
2. 「カテゴリ」ペインで、「Web Tier」の下の **「JSF」**をクリックして、Java Server Faces ページを作成します。
3. 「項目」で**「JSF JSP」**をクリックし、**「OK」**をクリックします。
4. **「次へ」**をクリックして、「ようこそ」ページをスキップします。
5. 「ファイル名」フィールドに `Welcome.jspx` を入力します。
6. 「タイプ」で**「JSP ドキュメント (*.jspx)」**が選択されていることを確認し、**「次へ」**をクリックします。
7. このページにはバックエンド・ロジックを追加するので、マネージド Bean が必要になります。ラジオ・ボタン**「新規マネージド Bean での UI コンポーネントの自動公開」**をクリックします。

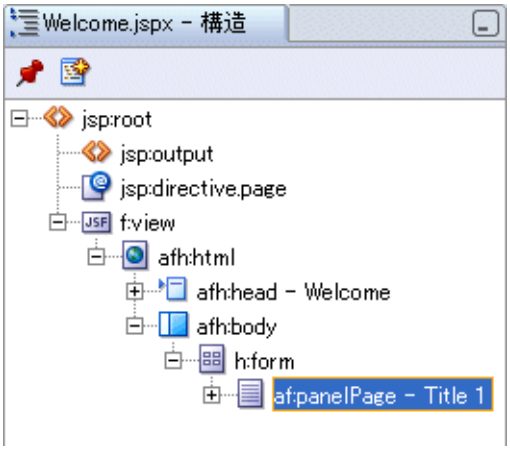
   「名前」、「クラス」および「パッケージ」は、示されているデフォルトのままにしておいてかまいません。
8. **「次へ」**をクリックします。
9. **「選択済のライブラリ」**ペインに次のライブラリが表示されていることを確認します。
   - ADF Faces Components 10_1_3_2_0
   - ADF Faces HTML 10_1_3_2_0
   - ADF Portlet Components 10_1_3_2_0
   - Customizable Components Core 10_1_3_2

- JSF Core 1.0
- JSF HTML 1.0

10. 「**次へ**」をクリックします。
11. 以前に、ログイン・ページに JDeveloper スタイルシートを適用しました。ようこそページにも JDeveloper スタイルシートを適用することにします。「**追加**」をクリックし、「**css**」フォルダをダブルクリックして「**jdeveloper.css**」を選択します。
12. 「**終了**」をクリックします。

    「設計」ビューに、Welcome.jspx という名前の空のページが表示されます。
13. 次に、ADF Faces の PanelPage コンポーネントを使用して、ようこそページにコンテンツを表示します。
    a. コンポーネント・パレットのプルダウンから、「**ADF Faces Core**」を選択します。
    b. 「**PanelPage**」オプションを選択し、ページにドラッグ・アンド・ドロップします。構造ウィンドウを使用して、PanelPage コンポーネントが「`h:form`」タグの中に配置されることを確認します（図 8-15 を参照）。

図 8-15 Welcome.jspx - ADF Faces の panelPage コンポーネント



    c. 構造ウィンドウで、「**af:panelPage - Title1**」をダブルクリックして、「PanelPage のプロパティ」ダイアログを表示します。
    d. 「タイトル」には、`Welcome to the Oracle WebCenter Framework Tutorial` を入力します。
    e. 「**OK**」をクリックします。

ようこそページは、図 8-16 のように表示されます。このチュートリアルでは、他の af:panelPage プロパティを設定する必要はありません。

図 8-16 新しい panelPage タイトルが表示された Welcome.jspx

今度は、PanelPage ファセットの 1 つである menuGlobal という名前の領域へのログイン・リンクおよびログアウト・リンクを追加します。

14. ログイン・リンクには、別の ADF Faces Core コンポーネント af:goLink を使用します。
    a. コンポーネント・パレットのプルダウンから、「**ADF Faces Core**」を選択します。
    b. 「**GoLink**」オプションを選択し、右上隅にある menuGlobal という名前の領域の上にドラッグ・アンド・ドロップします。構造ウィンドウを使用して、af:goLink コンポーネントが「menuGlobal」ファセットの中にあることを確認します（図 8-17 を参照）。

図 8-17 Welcome.jspx - ADF Faces の GoLink コンポーネント

c. 構造ウィンドウで、「**af:goLink - goLink1**」をダブルクリックして、「プロパティ」ダイアログを表示します。

d. リンクの「テキスト」に、`Login` を入力します。

e. 「宛先」フィールドに、次のように入力します。

```
/adfAuthentication?success_url=faces/Welcome.jspx
```

`adfAuthentication` サーブレットによって、ユーザーはログインするように要求されます。パラメータ `success_url` は、ログイン成功後に表示されるページ（このチュートリアルではようこそページ）を指定します。

f. 「**OK**」をクリックします。

図 8-18 のような「Login」ハイパーリンクが表示されます。

図 8-18 ようこそページの「Login」リンク

現在表示されているページ・コンテンツ（図 8-18）は、ようこそページのパブリック・ビューです。後で、いくつかのページ・ナビゲーション・ボタンを追加します。ただし、認証されていないユーザーに対してこれらのボタンが表示されないようにするためのコードを追加します。

同様に、認証されていないユーザーに対して「Login」リンクが表示されないようにするコードも追加します。もちろん、すでにログインしているユーザーにログイン・リンクを表示する必要はありません。それでは、現在のユーザーが認証されているかどうかを判別するバッキング Bean を追加し、これをログイン・リンクにバインドします。

15. まず、「**Welcome.java**」をダブルクリックして、エディタでファイルを開きます。

このファイルは、「**ViewController**」、「**アプリケーション・ソース**」、「**view.backing**」の下にあります。

16. 現在のユーザーがログインしているかどうかを判別するコードを追加します（図 8-19 で強調表示されているコードを参照）。

図 8-19 Welcome.java

```
package view.backing;

import javax.faces.component.html.HtmlForm;

import oracle.adf.share.ADFContext;
import oracle.adf.view.faces.component.core.layout.CorePanelPage;
import oracle.adf.view.faces.component.core.nav.CoreGoLink;
import oracle.adf.view.faces.component.html.HtmlBody;
import oracle.adf.view.faces.component.html.HtmlHead;
import oracle.adf.view.faces.component.html.HtmlHtml;

public class Welcome {

    private HtmlHtml html1;
    private HtmlHead head1;
    private HtmlBody body1;
    private HtmlForm form1;
    private CorePanelPage panelPage1;
    private CoreGoLink goLink1;
    private boolean authenticated;

    public void setHtml1(HtmlHtml html1) {...}

    public HtmlHtml getHtml1() {...}

    public CoreGoLink getGoLink1() {...}

    public boolean isAuthenticated() {
        authenticated =
                ADFContext.getCurrent().getSecurityContext().isAuthenticated();
        return authenticated;
    }
}
```

ここに示されている太字のコードをコピーして、Welcome.java の適切なセクションに貼り付けることができます。

```
package view.backing;

import javax.faces.component.html.HtmlForm;

import oracle.adf.share.ADFContext;
import oracle.adf.view.faces.component.core.layout.CorePanelPage;
import ...

public class Welcome {
 ...
 private CoreGoLink goLink1;
 private boolean authenticated;
 ...

 public CoreGoLink getGoLink1()
  {return goLink1;
  }
 public boolean isAuthenticated()
  {
  authenticated=ADFContext.getCurrent().getSecurityContext().isAuthenticated();
  return authenticated;
  }
}
```

現在のユーザーがログインしている場合は、バッキング Bean の boolean プロパティは TRUE です。ログインしていない場合は、FALSE です。このプロパティを使用して、ユーザーがログインしているかどうかに基づいて、ようこそページのリンクを表示または非表示にします。

17. 先に進む前に、Welcome.java への更新内容が正しくコンパイルされたことを確認します。「Welcome.java」を右クリックし、**「メイク」**を選択します。

    メッセージ・ログ・ウィンドウに、「コンパイルが成功しました」というメッセージが表示されます。

18. 次に、このコードを「Login」リンクにバインドします。「Login」リンクは認証されていないユーザーにのみ表示することを忘れないでください。

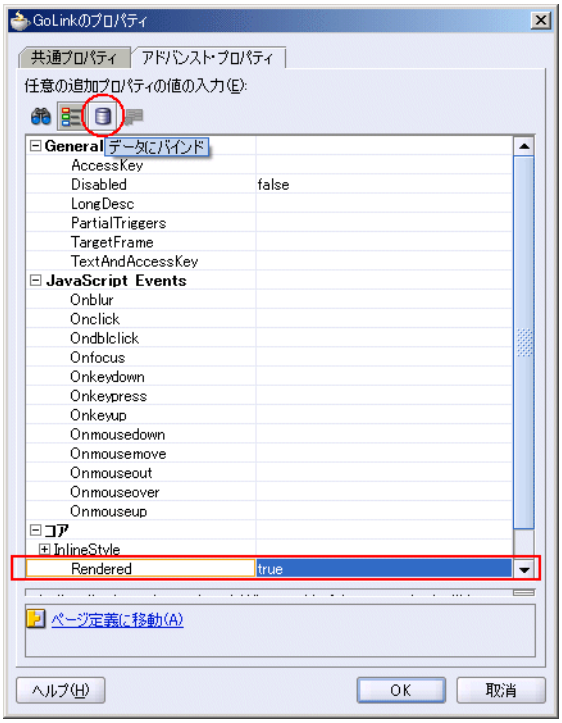
    a. アプリケーション・ナビゲータで、**「Welcome.jspx」**をダブルクリックして、ビジュアル・エディタでページを表示します。

    b. 構造ウィンドウで、**「af:goLink - Login」**をダブルクリックして、「GoLink のプロパティ」ダイアログを開きます。

    c. **「アドバンスト・プロパティ」**タブをクリックし、**「Rendered」**プロパティを選択します。

    d. ツールバーの**「データにバインド」**アイコンをクリックします（図 8-20 を参照）。

図 8-20 コマンド・ボタンの Rendered プロパティ



    e. **「JSF マネージド Bean」**を開き、ようこそページ **backing_Welcome** のバッキング Bean にドリルインします。この場所に、`authenticated` 属性が表示されます。

    f. **「authenticated」**をダブルクリックします。「式」フィールドに、`#{backing_Welcome.authenticated}` という式が表示されます。

    g. 認証されていないユーザーに「Login」リンクを表示することにします。ユーザーがログインしているときは、Authenticated プロパティは TRUE になるため、ユーザーがログインしていないときにのみリンクが表示されるように、式を否定する必要があります。`#{!backing_Welcome.authenticated}` のように、式に ! 演算子を追加します。

h. 「**OK**」をクリックします。

Rendered プロパティに、`#{!backing_Welcome.authenticated}` という式が表示されます。

i. 再度「**OK**」をクリックして、「GoLink のプロパティ」ダイアログを閉じます。

19. 今度は、認証されていないユーザーのみに表示されるログアウト・リンクを追加します。

a. コンポーネント・パレットのプルダウンから、「**ADF Faces Core**」を選択します。

b. 「**GoLink**」オプションを選択し、「Login」リンクの下（menuGlobal ファセットの中）にドラッグ・アンド・ドロップします。構造ウィンドウを使用して、これを確認します。

c. 構造ウィンドウで、「**af:goLink - goLink 1**」をダブルクリックして、「GoLink のプロパティ」ダイアログを開きます。

d. リンクの「テキスト」に、`Logout` を入力します。

e. 「宛先」フィールドに、次のように入力します。

```
/adfAuthentication?logout=true&end_url=faces/Welcome.jspx
```

`adfAuthentication` サーブレットによって、ユーザーはログアウトするように要求されます。パラメータ `end_url` は、ユーザーがログアウトした後に表示されるページを指定します。

f. 次に、認証されていないユーザーに対して「Logout」リンクを非表示にします。「**アドバンスト・プロパティ**」タブをクリックし、「**Rendered**」プロパティを選択します。

g. ツールバーの「**データにバインド**」アイコンをクリックします。

h. 認証済のユーザーにこのリンクを表示するには、「式」フィールドに次のように入力します。

```
#{backing_Welcome.authenticated}
```

i. 「**OK**」をクリックします。

j. 再度「**OK**」をクリックして、「プロパティ」ダイアログを閉じます。

ようこそページに「Logout」ハイパーリンクが表示されます（図 8-21）。

図 8-21 ようこそページの「Logout」リンク

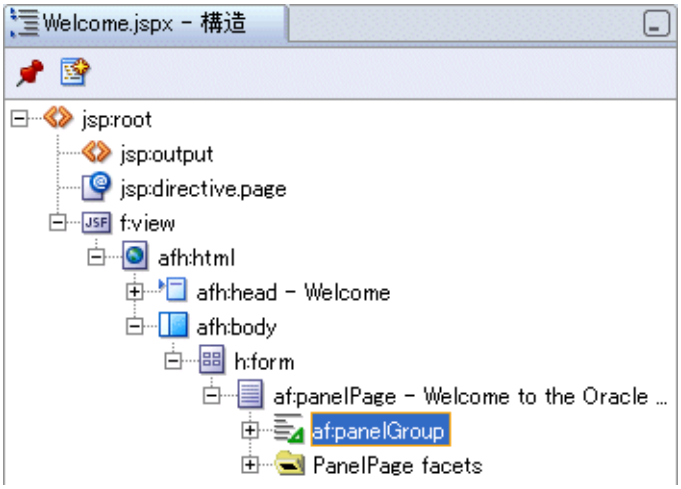
最後に、認証済のユーザーがチュートリアルのページ（MyPage.jspx、MyWeather.jspx および MyContent.jspx）にナビゲートできるように、ようこそページにいくつかのコマンド・ボタンを追加します。

20. ページ・ナビゲーション・ボタンを含めるためには、ADF Faces panelGroup コンポーネントを使用します。このコンポーネントを使用すると、コンテンツを水平または垂直に配置できます。

    a. コンポーネント・パレットのプルダウンから、「**ADF Faces Core**」を選択します。

    b. 「**PanelGroup**」オプションを選択し、「af:panelPage」コンポーネントの上にドラッグ・アンド・ドロップします。ここでも、構造ウィンドウを使用して、panelGroup コンポーネントが「af:panelPage」タグの中にあることを確認します（図 8-22 を参照）。

図 8-22 Welcome.jspx - ADF Faces の PanelGroup コンポーネント



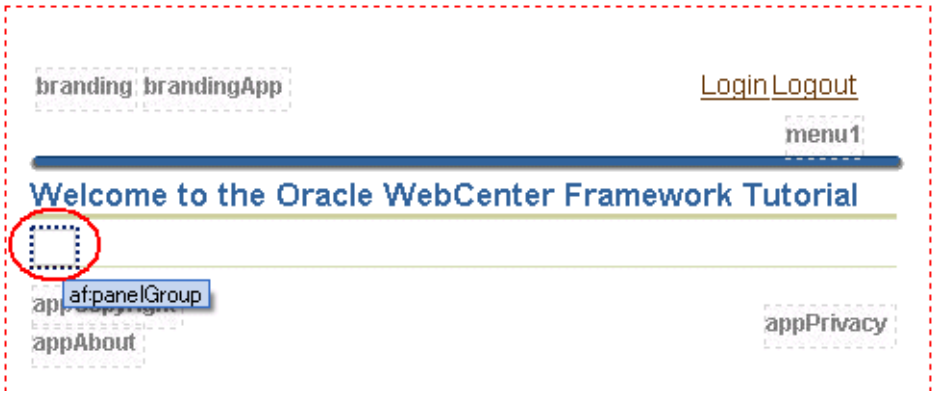
    c. 構造ウィンドウで、「**af:panelGroup**」をダブルクリックして、「プロパティ」ダイアログを表示します。

    d. 「レイアウト」で、「**水平方向**」を選択します。

    e. 「**OK**」をクリックします。

    ページ・タイトルの下に空のパネルが表示されます（図 8-23）。

図 8-23 Welcome.jspx - 空の panelGroup コンポーネント



21. Welcome.jspx から MyPage.jspx にナビゲートするコマンド・ボタンを追加する前に、2 つのページ間の JSF ナビゲーション・ルールを作成する必要があります。その後は、ADF コマンド・ボタンをページにドロップし、認証済ユーザーにのみボタンを表示するボタン・コードを追加できます。

    a. まず、Welcome.jspx と MyPage.jspx の間のナビゲーション・ルールを定義します。これは、構成ファイル `faces-config.xml` 内で定義します。アプリケーション・ナビゲータで、(「ViewController」、「Web コンテンツ」、「WEB-INF」の下の)**「faces-config.xml」**をダブルクリックします。

    b. **「ダイアグラム」**タブをクリックします。

    c. アプリケーション・ナビゲータから**「Welcome.jspx」**を空のダイアグラムにドラッグし、次に**「MyPage.jspx」**に対しても同じことを行います(図 8-24)。

図 8-24 JSF ナビゲーション・ダイアグラム

コンポーネント・パレットで JSF Navigation Modeler のコンポーネントが自動的に表示されることに注意してください。

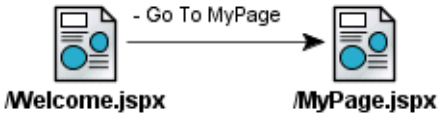
    d. コンポーネント・パレットから、**「JSF ナビゲーション・ケース」**を選択してアクティブにします。このコンポーネントを使用して、ナビゲーション・ルールを追加します。

    e. ダイアグラムで、ソース・ページ(**Welcome.jspx**)のアイコンをクリックしてから、宛先ページ(**MyPage.jspx**)のアイコンをクリックします。

    JDeveloper によって、ダイアグラムにナビゲーション・ルールが矢印付きの実線で描かれます。

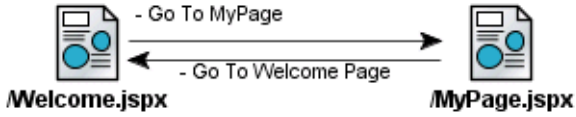
    f. デフォルトでは、リンクの結果に `-success` と表示されます。このテキストをクリックし、`Go To MyPage` のような、より説明的なテキストに変更します(図 8-25)。

図 8-25 ルールを持つ JSF ナビゲーション・ダイアグラム



    g. 今度は、同様の手順に従って、MyPage.jspx から Welcome.jspx へのナビゲーション・ルールを作成します。まず、コンポーネント・パレットから**「JSF ナビゲーション・ケース」**を選択します。次に、(ダイアグラムの)**「MyPage.jspx」**をクリックし、続いて**「Welcome.jspx」**をクリックします。最後に、デフォルトの `-success` テキストをクリックして `Go To Welcome Page` に変更します(図 8-26)。

図 8-26 2 つのルールを持つ JSF ナビゲーション・ダイアグラム



これで 2 つのページ間にナビゲーション・ルールが作成されたので、ようこそページ上に、MyPage へのユーザー・ナビゲーション用のコマンド・ボタンを追加します。

h. アプリケーション・ナビゲータで、**「Welcome.jspx」**をダブルクリックして、ビジュアル・エディタでこれを表示します。

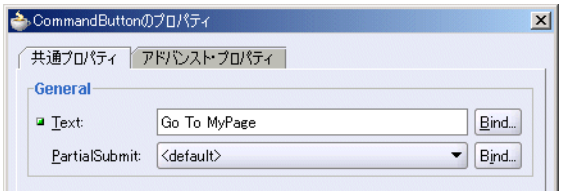
i. コンポーネント・パレットのプルダウン・メニューから、**「ADF Faces Core」**を選択します。

j. **「CommandButton」**オプションを選択し、「af:panelGroup」の上にドラッグ・アンド・ドロップします。構造ウィンドウを使用して、これを確認します。

k. 新しいボタン（「commandButton 1」）を右クリックし、**「プロパティ」**を選択します。

l. ボタンの「テキスト」に、Go To MyPage を入力し（図 8-27 を参照）、**「OK」**をクリックします。

図 8-27 Welcome Page.jspx - ページ・ナビゲーション・ボタンの構成



m. ボタンの実際の動作は、ボタンの Action プロパティを介して定義します。これを行うにはプロパティ・インスペクタを使用する必要があるので、JDeveloper メニューから**「表示」**、**「プロパティ・インスペクタ」**を選択します。

n. プロパティ・インスペクタで、**「Action」**を選択し、ドロップダウン・リストから**「MyPage に移動」**を選択します。

次に、「Logout」リンクに行ったように、認証されていないユーザーに対してボタンを非表示にします。また、ログインしているユーザーに MyPage.jspx の表示権限がない場合にボタンを非表示にするコードを追加します。今度は、プロパティ・インスペクタを使用して式を入力します。

o. **Rendered** プロパティを選択します。プロパティ・インスペクタでは、このプロパティは**「コア」**セクションの下にあります。

p. 次に、インスペクタのツールバーで**「データにバインド」**アイコンをクリックします。

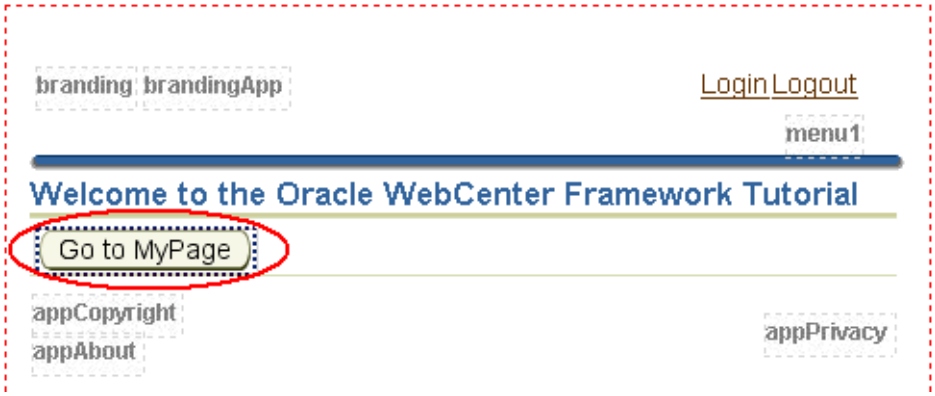
q. `#{backing_Welcome.authenticated && bindings.permissionInfo['MyPagePageDef'].allowsView}` という式を入力します。

この式の先頭または末尾に余分な空白がないことを確認してください。

r. 「**OK**」をクリックします。

JDeveloper で、ようこそページは図 8-28 のように表示されます。

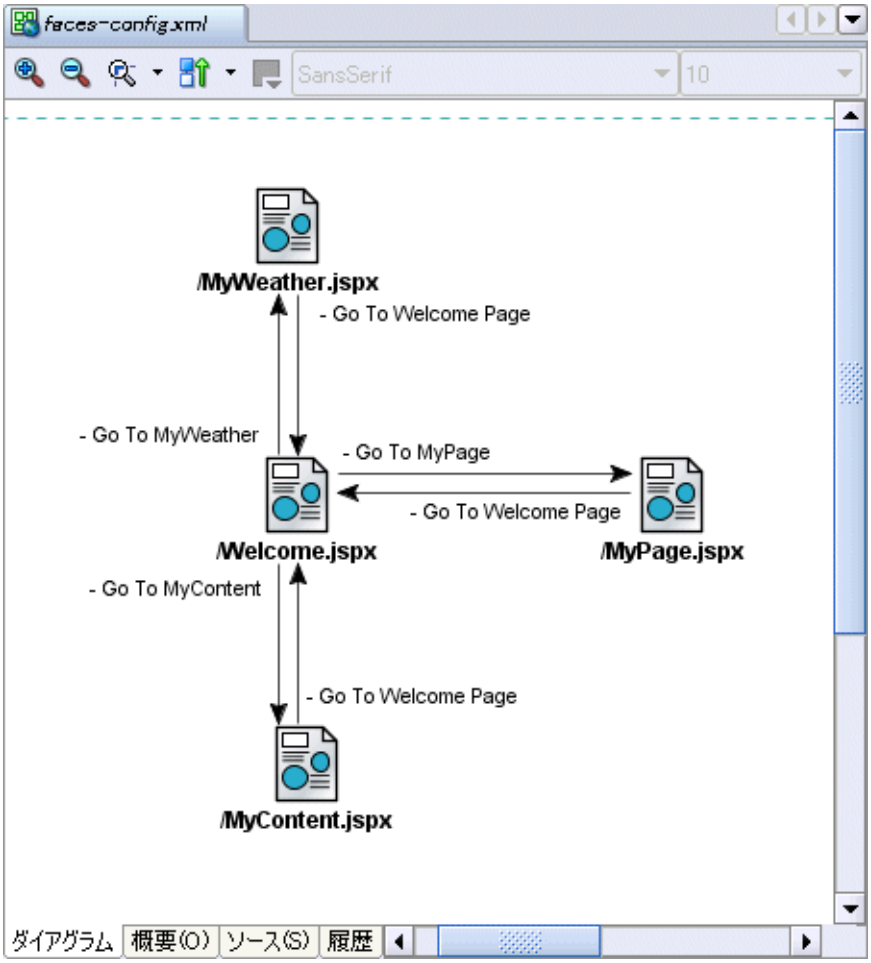
図 8-28 ようこそページの MyPage ナビゲーション・ボタン



22. 今度はさらに、MyWeather.jspx および MyContent.jspx への認可済ユーザーのナビゲーションを可能にする 2 つのボタンをようこそページに追加します。手順 21 を、慎重に繰り返してください。

    a. まず、MyWeather.jspx および MyContent.jspx をナビゲーション・ダイアグラムに追加します。完了すると、ナビゲーション・ダイアグラムは図 8-29 のように表示されます。

図 8-29 完了後の JSF ナビゲーション・ダイアグラム



    b. 次に、さらに 2 つのボタンを Welcome.jspx に追加し、ボタンの **Text**、**Action** および **Rendered** プロパティを設定します。

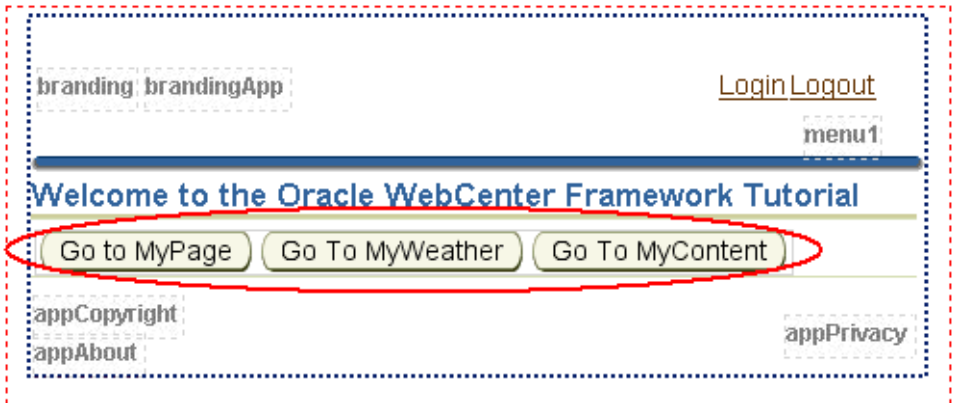
    これらのナビゲーション・ボタンに対する **Text**、**Action** および **Rendered** プロパティを編集するときは、忘れずに、（表 8-1 に示す）適切なページ名に置き換え、Rendered 式の最初と最後に余分な空白がないかを確認してください。

表 8-1 コマンド・ボタンのプロパティ

ボタンの Text	Action	Rendered
Go To MyWeather	Go To MyWeather	#{backing_Welcome.authenticated && bindings.permissionInfo['**MyWeatherPageDef**'].allowsView}
Go To MyContent	Go To MyContent	#{backing_Welcome.authenticated && bindings.permissionInfo['**MyContentPageDef**'].allowsView}

ようこそページは、図 8-30 のように表示されます。

図 8-30 最終的なようこそページ



23. JDeveloper のツールバーで、**「すべて保存」**アイコンをクリックします。

    次に、ようこそページを実行して、追加したリンクおよびボタンが予測どおりに表示されるかどうかを調べます。

24. **「Welcome.jspx」**を右クリックし、**「実行」**を選択します。

    ブラウザで、「Login」リンクを備えたようこそページが表示されます（図 8-31 を参照）。認証されていない場合、ログアウト・リンクとページ・ナビゲーション・ボタンは表示されません。

    ページがこのように表示されない場合は、予測どおりに表示されていないリンクまたはボタンの Rendered 式を確認してください。

図 8-31 ようこそページ - パブリック・ビュー

25. ブラウザを閉じ、JDeveloper に戻ります。

次の手順では、アプリケーション内の各ページ（Welcome.jspx、MyPage.jspx、MyWeather.jspx および MyContent.jspx）に対してアクセス権限を認可します。

手順 4: ページの保護

この手順では、このチュートリアル・アプリケーション内のページ（Welcome.jspx、MyPage.jspx、MyWeather.jspx および MyContent.jspx）を保護（アクセスを制限）します。ID ストア内に定義されているロール・メンバー（「前提条件」を参照）へのページ・アクセスを制限し、ページ上でロール・メンバーが実行できるアクションを決定します。保護されたページにもログアウト・リンクが必要なので、各ページの一番上にログアウト・リンクも追加します。

まずは、ようこそページから開始します。このページにはすでにログアウト・リンクが備わっていますが、さらに、ページへのアクセスを認可し、許可されるアクションを指定する必要があります。この構成は、ページの定義ファイル（`WelcomePageDef.xml`）で行います。それでは、始めましょう。

1. アプリケーション・ナビゲータで、**「Welcome.jspx」**を右クリックします。
2. **「ページ定義に移動」**を選択します。

   ページ定義がまだ存在していない場合は、**「はい」**をクリックして、Welcome.jspx のページ定義を作成します。
3. 構造ウィンドウで、**「WelcomePageDef」**を右クリックし、**「認可の編集」**を選択します。

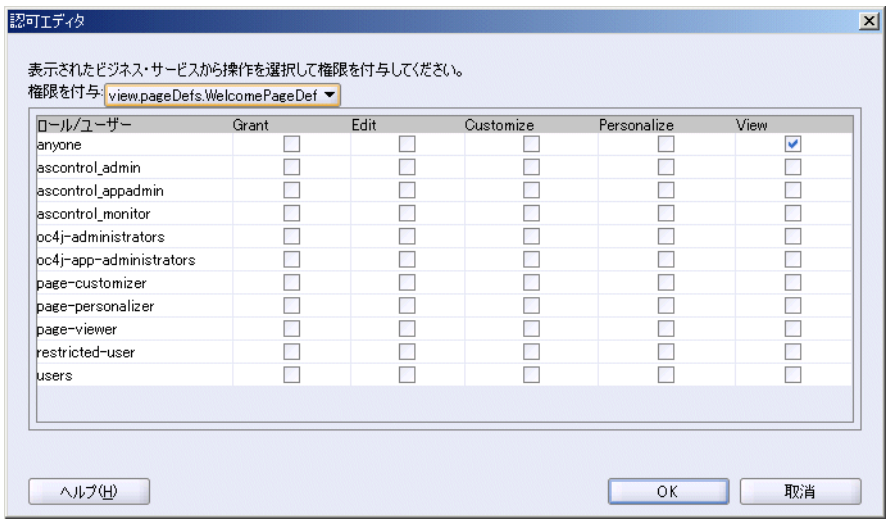
   認可エディタに、ID ストアのロールがリストされます（図 8-32 を参照）。これらのロールが表示されない場合は、埋込み OC4J ディレクトリ `JDEVHOME¥jdev¥system¥oracle.j2ee.10.1.3.xx.xx¥embedded-oc4j¥config` からディレクトリ `JDEVHOME¥j2ee¥home¥config` にファイル `system-jazn-data.xml` をコピーしたことを確認してください。これは、この章の前提条件の 1 つとなっています。

   認可エディタでは、各ロールが実行できるページ・アクションを選択することもできます。

   - **Grant:** ユーザーはページ権限を管理（権限付与 / 取消し）できます。
   - **Edit:** ユーザーは、ページに表示されたコンテンツを編集できます。このリリースでは、「Edit」アクションは適用できません。
   - **Customize:** ユーザーはページを変更できます。この権限が付与されていないユーザーは、ページを変更できません。
   - **Personalize:** ユーザーは、ページ上のポートレットをパーソナライズできます。ユーザーにこの権限が付与されていない場合、ページ・ポートレットをパーソナライズ・モードにするリンクまたはボタンは表示されません。
   - **View:** ユーザーはページを表示できます。この権限を付与されていないユーザーには、認証エラーが表示されます。

4. Welcome.jspx はすべての人に表示されます。**anyone** ロールに対して「**View**」チェック・ボックスを選択します（図 8-32 を参照）。

図 8-32 Welcome.jspx - 認可エディタ



5. 「**OK**」をクリックします。

   認可エディタを介して行った変更は、すぐにテストできるように、埋込み OC4J の `system-jazn-data.xml` ファイルに保存されます。また同時に、アプリケーションの `.adf/META-INF` ディレクトリ内の `app-jazn-data.xml` にも保存されます。`app-jazn-data.xml` ファイルは、アプリケーション固有のセキュリティ・ポリシーをアプリケーション自体とともにパッケージ化して、セキュアな WebCenter アプリケーションのデプロイを促進します。`app-jazn-data.xml` については、第 9 章「WebCenter アプリケーションのデプロイ」で詳しく学びます。

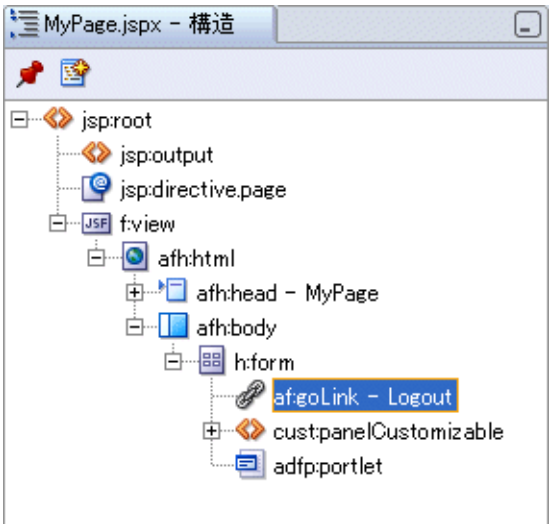
   次に、MyPage.jspx を保護します。まずログアウト・リンクを追加し、続いてページ認可詳細を編集します。新しいログアウト・リンクを最初から作成するのではなく、ようこそページに作成したログアウト・リンクをコピーします。

6. アプリケーション・ナビゲータで、「**Welcome.jspx**」を選択します。
7. 構造ウィンドウで、「**af:goLink - Logout**」を右クリックして、「**コピー**」を選択します。
8. アプリケーション・ナビゲータで、「**MyPage.jspx**」を選択します。

9. 構造ウィンドウで、「**h:form**」を選択し、右クリックして「**貼付け**」を選択します。

   デフォルトで、ページの一番下に「`af:goLink - Logout`」が配置されます。これを「`cust:panelCustomizable`」の上にドラッグして、ページの一番上に移動します（図 8-33）。

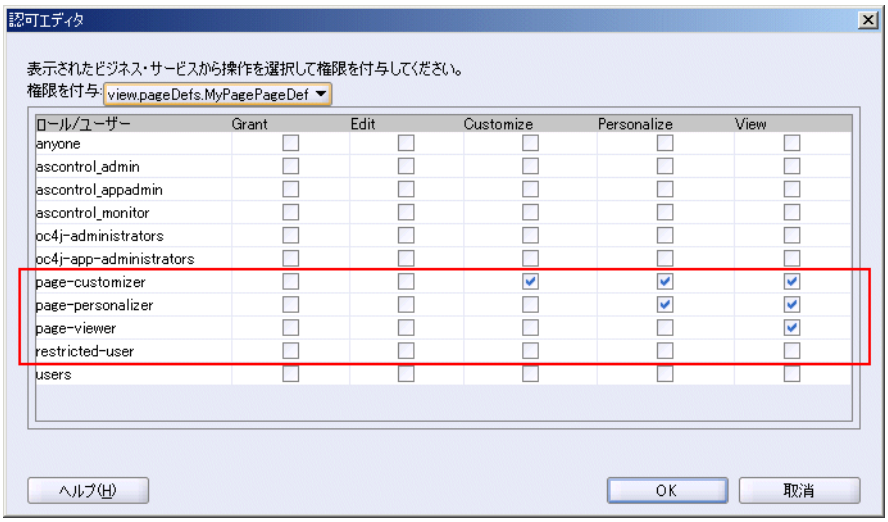
図 8-33 MyPage.jspx - 構造ウィンドウ内のログアウト・リンク



   今度は、このページへのアクセスを認可し、ロールごとにロールに対して許容されるアクションを指定します。前と同様に、この構成はページの定義ファイル（`MyPagePageDef.xml`）で行います。

10. アプリケーション・ナビゲータで、「**MyPage.jspx**」を右クリックします。
11. 「**ページ定義に移動**」を選択します。
12. 構造ウィンドウで、「**MyPagePageDef**」を右クリックし、「**認可の編集**」を選択します。
13. 認可エディタを使用して、MyPage.jspx 上のユーザー権限を付与します。図 8-34 に示されたチェック・ボックスを選択します。

図 8-34 MyPage.jspx - 認可エディタ



これらの設定について詳しく説明します。*page-customizer* ロールが割り当てられたユーザー（*Harvey* など）は、チュートリアル・アプリケーションにログインすると、MyPage.jspx の状況の表示、パーソナライズおよびカスタマイズはできますが、ページ・コンテンツを編集したり、他のユーザーにページ権限を付与することはできません。*page-viewer* ロールを持つログイン済ユーザー（*Singh* など）は、さらに制限が多くなります。これらのユーザーは MyPage.jspx を表示できますが、それ以外のページ・アクションは許可されません。

restricted-user ロールを持つユーザー（*King* など）は、ページの表示権限をまったく持ちません。

14. **「OK」**をクリックして、これらの選択内容を保存します。

    これで MyPage.jspx がセキュアになったので、MyWeather および MyContent にもまったく同じ手順を繰り返します。

15. MyWeather および MyContent を保護するには、手順 8 ～ 14 を繰り返します。

    ログアウト・リンクをコピーして貼り付けた後は、忘れずにリンクをページの一番上（「`h:form`」の下）に移動してください。

16. JDeveloper のツールバーで、**「すべて保存」**アイコンをクリックします。

    これで、アプリケーション内のすべてのページがセキュアになりました。

手順 5: orion-web.xml でのセキュリティ・ロールのマッピング

チュートリアル・アプリケーションを保護するための手順が、もう 1 つあります。Oracle ADF セキュリティ・ウィザードを使用して定義した J2EE セキュリティ・ロール（`ValidUsers`）を、`system-jazn-data.xml` ファイル内に定義されている ID ストア・ロール（`users`）にマップする必要があります。

J2EE セキュリティ・ロール	ID ストア・ロール
ValidUsers	users

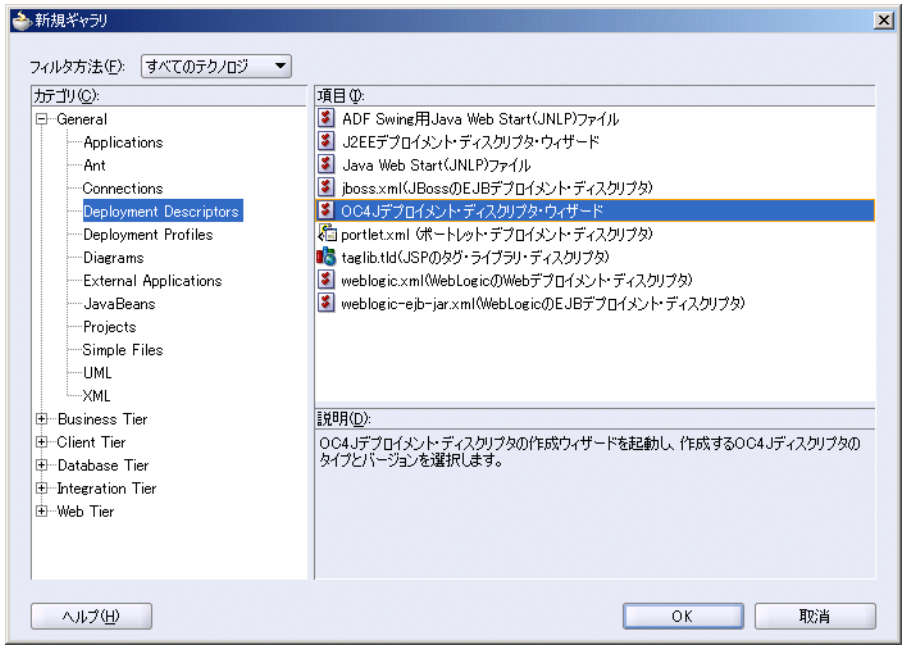
まずは、このようなセキュリティ・ロール・マッピングを格納する構成ファイル、つまり、チュートリアル・アプリケーションの OC4J デプロイメント・ディスクリプタ・ファイル（`orion-web.xml`）を作成します。それでは、このファイルを作成しましょう。

1. アプリケーション・ナビゲータで、**「ViewController」**を右クリックし、**「新規」**を選択します。
2. 一番上のプルダウン・リストから、**「すべてのテクノロジ」**を選択します。

3. 左側のパネルで、「**General**」を開き、「**Deployment Descriptors**」を選択します（図 8-35 を参照）。右側のパネルには、様々なタイプの使用可能なデプロイメント・ディスクリプタがリストされます。プロジェクトで使用するテクノロジや、ターゲット・アプリケーション・サーバーのタイプに応じて、様々なデプロイメント・ディスクリプタが必要になります。

   このチュートリアル・アプリケーションは、Preconfigured OC4J サーバー上にデプロイされます。

図 8-35 デプロイメント・ディスクリプタの選択



4. 「**OC4J デプロイメント・ディスクリプタ・ウィザード**」を選択し、「**OK**」をクリックします（図 8-35）。

   これによって、OC4J デプロイメント・ディスクリプタ・ウィザードが起動されます。このウィザードでは、必要な特定のデプロイメント・ディスクリプタのタイプおよびバージョンを選択し、.xml ファイルを生成できます。このチュートリアルでは、デプロイメント・ディスクリプタとして「`orion-web.xml 10.0`」を選択する必要があります。後で、アプリケーション・サーバーにデプロイする前に、この .xml ファイルを使用してチュートリアル・アプリケーションをパッケージ化します。

5. 「**次へ**」をクリックして、「ようこそ」ページの先に進みます。

6. （リストの一番下にある）「**orion-web.xml**」を選択し、「**次へ**」をクリックします。

   ファイルがすでに存在する場合は、ファイル名がグレー表示されます。

7. 「**10.0**」を選択し、「**終了**」をクリックします。

   「終了」をクリックすると、アプリケーション・ナビゲータで「**ViewController**」、「**Web コンテンツ**」、「**WEB-INF**」の下に orion-web.xml が表示されます。

8. 「**orion-web.xml**」を右クリックし、「**プロパティ**」を選択して、追加のデプロイメント・オプションを設定します。

9. 左側のパネルから、**「セキュリティ・ロール・マッピング」**を選択します。

   これによって、右側にパネルが表示されます。このパネルに、次のセキュリティ・ロール・マッピングを追加します。

J2EE セキュリティ・ロール	ID ストア・ロール
ValidUsers	users

10. 次のようにして、セキュリティ・ロール・マッピングを作成します。

    a. **「追加」**をクリックします。

       これによって、図 8-36 に示すようなウィンドウが表示されます。このウィンドウに、J2EE セキュリティ・ロール名（`ValidUsers`）を入力します。

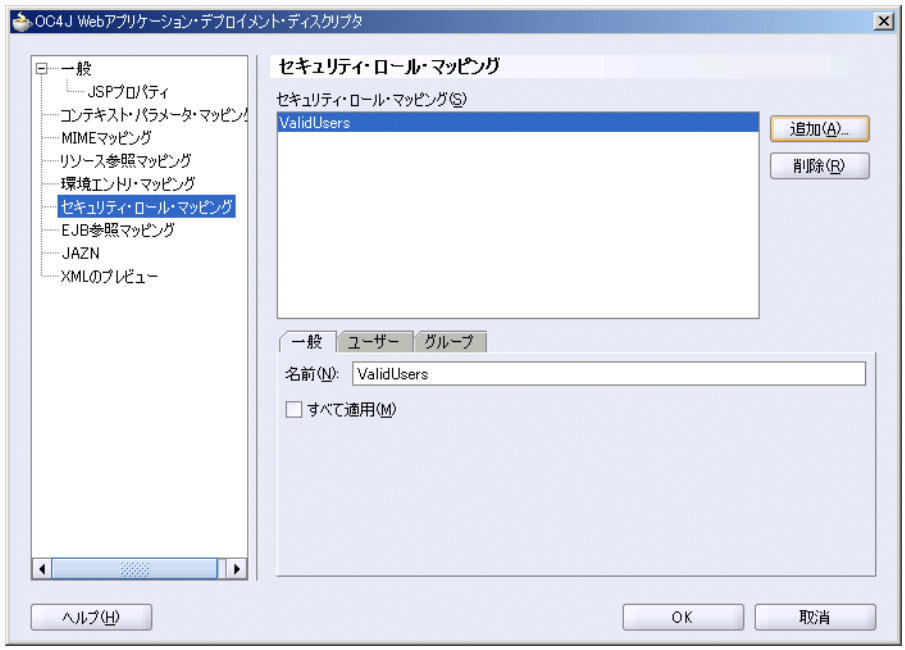
図 8-36 デプロイメント・ディスクリプタ - J2EE セキュリティ・ロール・マッピングの作成

    b. 「名前」には、J2EE セキュリティ・ロール名 `ValidUsers` を入力します。

c. 「**OK**」をクリックします。

入力したロール名が、マッピング・パネル内と「**一般**」タブに表示されます（図 8-37）。ロール名を編集する必要がある場合は、「一般」タブの「**名前**」プロパティを編集します。

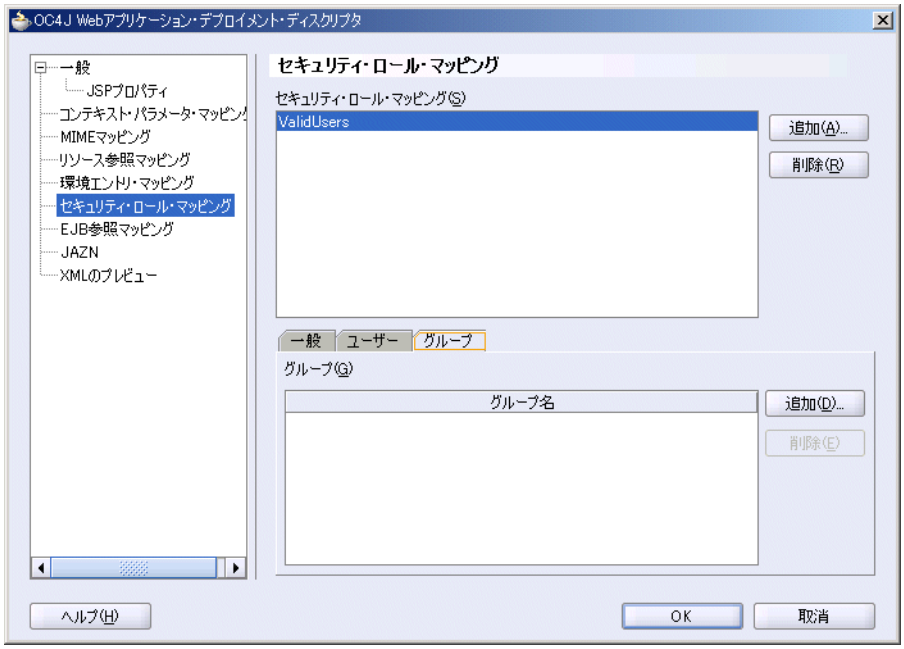
図 8-37 デプロイメント・ディスクリプタ - 新しい J2EE セキュリティ・ロール・マッピング



d. 「**グループ**」タブをクリックします。

マッピング・パネル内で J2EE セキュリティ・ロール ValidUsers が強調表示されていることに注意してください。これは、users グループをこの J2EE セキュリティ・ロールにマップするということです（図 8-38）。

図 8-38 デプロイメント・ディスクリプタ - 「セキュリティ・ロール・マッピング」の「グループ」タブ

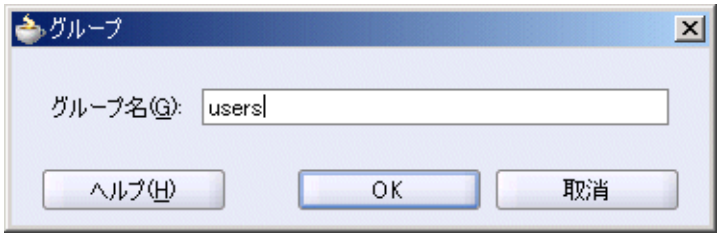


e. 「グループ名」パネルの右側にある「**追加**」ボタンをクリックします。

f. 「グループ名」に `users` を入力します（図 8-39）。

これは、有効な各ユーザー（Singh、Cho、Harvey、JtaAdmin および oc4jadmin）のリストを保守するデフォルトの ID ストア・ロールです。詳細は、付録 A「チュートリアルの ID ストアの設定方法」を参照してください。

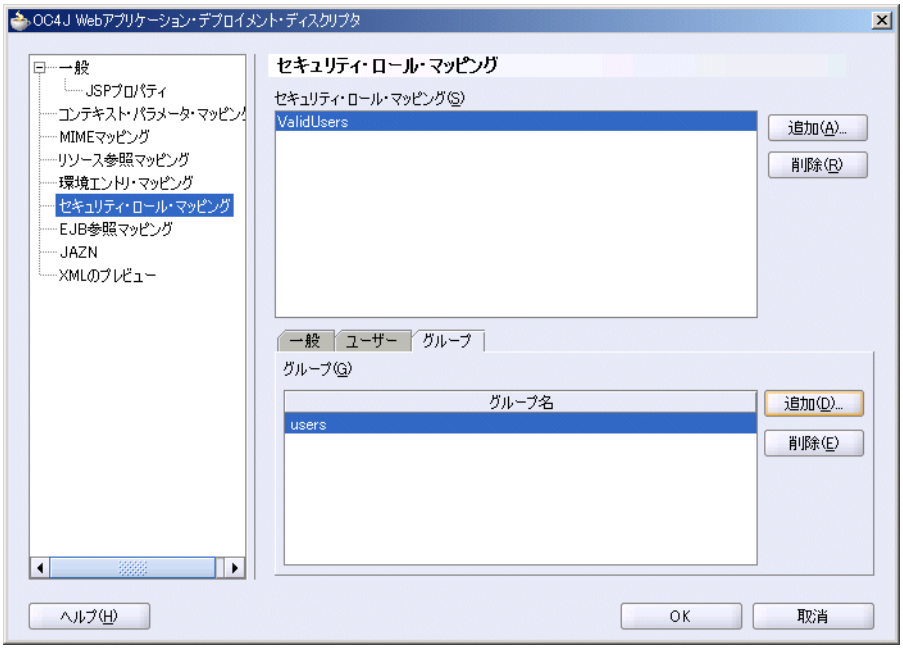
図 8-39 デプロイメント・ディスクリプタ - J2EE セキュリティ・ロールへの users グループのマッピング



g. 「**OK**」をクリックします。

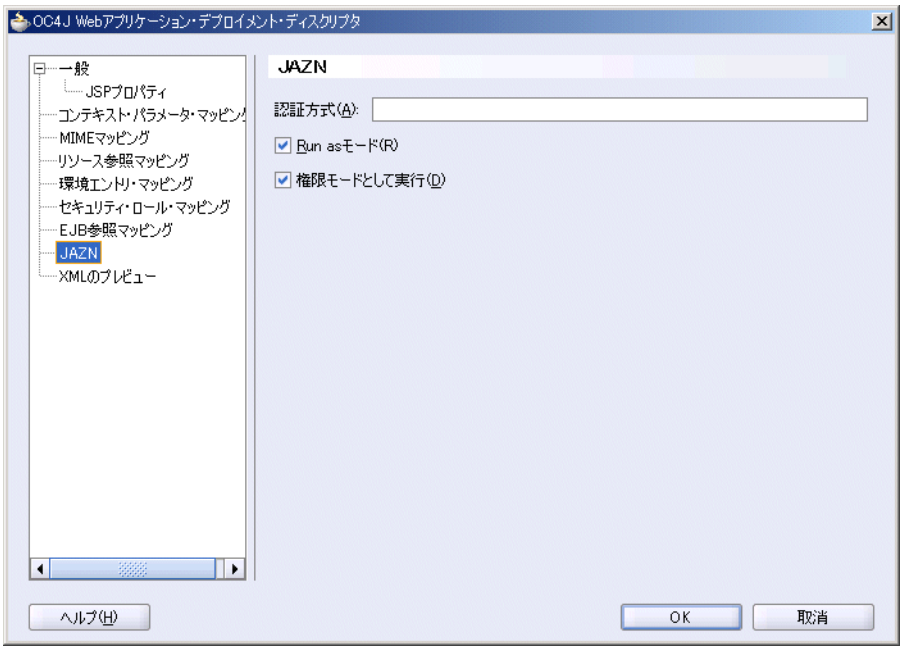
この手順では、J2EE セキュリティ・ロール ValidUsers を ID ストア・ロール users にマップしました（図 8-40）。

図 8-40 デプロイメント・ディスクリプタ - users ロールにマップされた J2EE セキュリティ・ロール



11. 左側のパネルで「**JAZN**」を選択し、「**Run as モード**」および「**権限モードとして実行**」を選択します（図 8-41 を参照）。

図 8-41 デプロイメント・ディスクリプタ - JAZN 設定



12. 「**OK**」をクリックして、OC4J デプロイメント・ディスクリプタへの変更を保存します。
13. JDeveloper のツールバーで、「**すべて保存**」アイコンをクリックします。

構成ファイル orion-web.xml のソース・コードを調べると、security-role-mapping エントリおよび jazn-web-app エントリは次のようになっています。

```
<security-role-mapping name="ValidUsers" impliesAll="false">
    <group name="users"></group>
 </security-role-mapping>
<jazn-web-app runas-mode="true" doasprivileged-mode="true"/>
```

これで、OC4J Web アプリケーションのデプロイメント・ディスクリプタの構成は完了です。アプリケーションを実行し、新しいセキュリティ機能の動作を確認できます。

手順 6: セキュリティ機能のデモ

この最後の手順では、チュートリアル・アプリケーションを実行し、導入したセキュリティ機能を調べます。これから行う手順の概要を簡単に示します。

- ようこそページ（Welcome.jspx）を表示し、ログイン・リンクによりログイン・ページ（Login.jspx）が表示されることをテストします。
- 「Login」ページを使用して、有効なユーザー資格証明を入力し、適切なページ（Welcome.jspx）が表示されることを確認します。ログインしているユーザーには、ようこそページにログイン・リンクは表示されず、ログアウト・リンクが表示されます。また、ユーザーに必要なページ権限があれば、MyPage、MyContent および MyWeather にナビゲートするボタンも表示されます。
- 様々なユーザー資格証明（Singh、Cho、Harvey および King）を使用してログインし、各ユーザーに実行が許可されているページ・アクションを確認します。
- 事前の認可なしでセキュア・ページへの直接のアクセスを試行し、「Login」ページにリダイレクトされることを確認します。
- 無効なユーザー資格証明を入力し、適切なエラー・メッセージが表示されることを確認します。

それでは、チュートリアル・アプリケーションを実行し、これらのセキュリティ機能の動作を確認します。

ユーザー Singh としてのログイン

まずは、ユーザー Singh としてログインします。このユーザーには、MyPage、MyWeather および MyContent に対する表示権限が割り当てられています。

1. アプリケーション・ナビゲータで、**「Welcome.jspx」**を右クリックし、**「実行」**を選択します。

   ブラウザで、「Login」リンクを備えたようこそページが表示されます（図 8-42 を参照）。このリンクは、ログイン・ページ Login.jspx が表示されるように以前に構成したものです。

図 8-42 ようこそページ - パブリック・ビュー

2. **「Login」**をクリックします。

   ユーザー資格証明の入力フォームを含む Login.jspx が表示されます（図 8-43 を参照）。

3. ユーザー Singh のログイン資格証明を入力します。名前とパスワードではいずれも大 / 小文字が区別されるため、入力する際には注意してください。「Name」には `Singh` を、「Password」には `welcome` を入力します。

図 8-43「Login」ページ - ユーザー Singh のログイン資格証明

4. **「Login」**をクリックします。

   認証が成功すると、ページの一番上に「Logout」リンクを備えた Welcome.jspx が表示されます（図 8-44 を参照）。ようこそページに「Login」リンクを構成したとき、Welcome.jspx を成功 URL として定義したことを思い出してください。

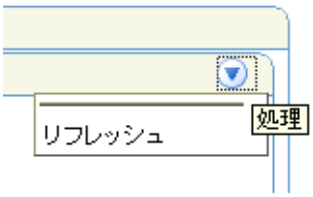
図 8-44 ようこそページ - 保護されたビュー

5. また、ユーザー Singh には、複数のコマンド・ボタンが表示されます。**Go To MyPage** というラベルのボタンをクリックします。

   ユーザー Singh は、このページを表示する権限がありますが、ページのパーソナライズまたはカスタマイズはできないはずです。これを確認するには、MyJavaPortlet のアクション・メニューを調べます。

6. ポートレット・バナーの**アクション**・アイコンをクリックすると、選択可能なオプションが表示されます。（表示権限を持っている）Singh としてログインしているため、「リフレッシュ」オプションのみが選択可能になります（図 8-45 を参照）。

図 8-45 ユーザー Singh に選択可能なポートレット・アクション



7. **「Logout」**をクリックします。

再びようこそページが表示されます。

ユーザー Cho としてのログイン

次は、ユーザー Cho としてログインします。このユーザーには、MyPage、MyWeather および MyContent に対する表示権限とパーソナライズ権限の両方を持つ page-personalizer ロールが割り当てられています。

1. ようこそページを表示し、**「Login」**をクリックします。
2. ユーザー Cho のログイン資格証明を入力します。両方のフィールドで大 / 小文字が区別されることを忘れないでください。「Name」には `Cho` を、「Password」には `welcome` を入力します（図 8-46）。

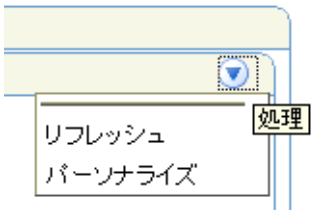
図 8-46「Login」ページ - ユーザー Cho のログイン資格証明

3. **「Login」**をクリックします。

   認証が成功すると、ページの一番上の「Logout」リンクおよび 3 つのページ・ナビゲーション・ボタンすべてを備えたようこそページが表示されます。
4. **「Go To MyPage」**をクリックします。

   ユーザー Cho は、ポートレットの表示権限とパーソナライズ権限を持ちますが、ページまたはポートレットのカスタマイズはできないはずです。これを確認するには、MyJavaPortlet のアクション・メニューを調べます。
5. ポートレット・バナーの**アクション**・アイコンをクリックすると、選択可能なオプションが表示されます。Cho としてログインしているため、「リフレッシュ」と「パーソナライズ」の 2 つのオプションが表示されます（図 8-47 を参照）。

図 8-47 ユーザー Cho に選択可能なポートレット・アクション



6. **「パーソナライズ」**をクリックします。
7. 「ポートレット・タイトル」を変更します。たとえば、`Cho's Java Portlet` と入力します。

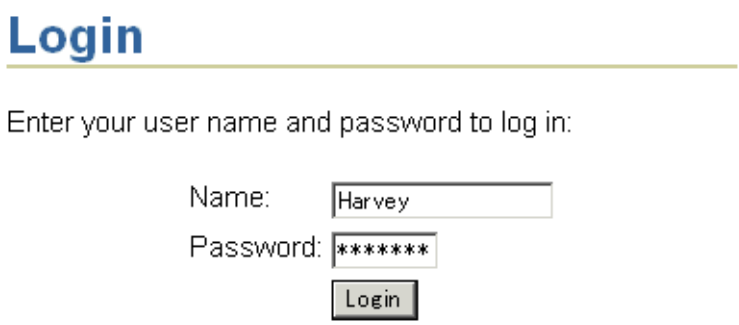
   このタイトルは、Cho がログオン・ユーザーの場合のみに表示されます。他のユーザーにはこのタイトルは表示されません。
8. **「OK」**をクリックして、ユーザー Cho のパーソナライズ内容を確認します。
9. **「Logout」**をクリックします。

ユーザー Harvey としてのログイン

今度は、ユーザー Harvey としてログインします。このユーザーには、MyPage、MyWeather および MyContent に対する表示権限、パーソナライズ権限およびカスタマイズ権限を持つ page-customizer ロールが割り当てられています。

1. ようこそページを表示し、**「Login」**をクリックします。
2. ユーザー Harvey のログイン資格証明を入力します。両方のフィールドで大 / 小文字が区別されることを忘れないでください。「Name」には `Harvey` を、「Password」には `welcome` を入力します（図 8-48）。

図 8-48「Login」ページ - ユーザー Harvey のログイン資格証明



3. **「Login」**をクリックします。

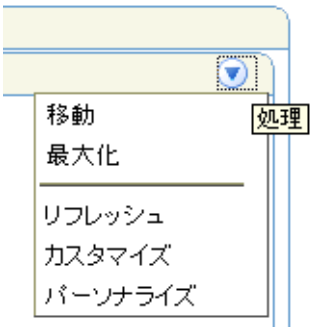
   認証が成功すると、ページの一番上の「Logout」リンクおよび前と同様の複数のページ・ナビゲーション・ボタンを備えた Welcome.jspx が表示されます。

4. **「Go To MyPage」**をクリックします。

   ユーザー Harvey は、表示権限、ポートレットのパーソナライズ権限、およびこのページのカスタマイズ権限を持ちます。これを確認するには、MyJavaPortlet のアクション・メニューを調べます。

5. ポートレット・バナーの**アクション**・アイコンをクリックすると、選択可能なオプションが表示されます。Harvey としてログインしているため、「移動」、「最大化」、「リフレッシュ」、「カスタマイズ」および「パーソナライズ」の 5 つのオプションが表示されます（図 8-49 を参照）。

図 8-49 ユーザー Harvey に選択可能なポートレット・アクション



ユーザー Harvey は、このページ上のコンテンツをカスタマイズまたは移動できます。そしてこれを行うと、すべてのユーザーのページを変更することになります。これは、変更を行うユーザーのみに適用されるパーソナライズとは異なります。先にユーザー Cho が行ったポートレット・タイトルのパーソナライズ内容は、ユーザー Harvey には表示されないことに注意してください。

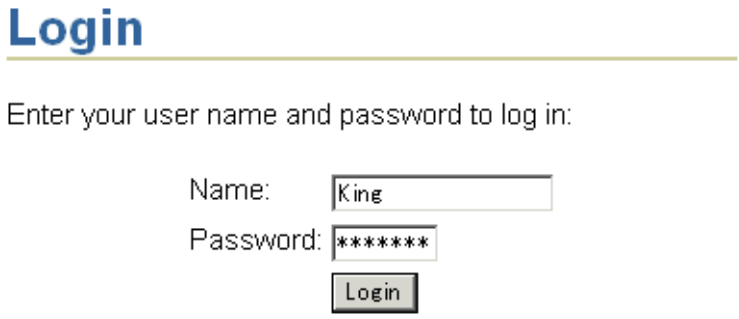
6. 「**Logout**」をクリックします。

ユーザー King としてのログイン

最後は、ユーザー King としてログインします。このユーザーには restricted-user ロールが割り当てられているため、パブリック・コンテンツにしかアクセスできません。このユーザーは、保護されたページ（MyPage、MyWeather または MyContent）を表示できません。

1. ようこそページを表示し、「**Login**」をクリックします。
2. ユーザー King のログイン資格証明を入力します。両方のフィールドで大 / 小文字が区別されることを忘れないでください。「Name」には `King` を、「Password」には `welcome` を入力します（図 8-50）。

図 8-50「Login」ページ - ユーザー King のログイン資格証明



3. 「**Login**」をクリックします。

   認証が成功すると、Welcome.jspx は表示されますが、ページ・ナビゲーション・ボタンは表示されません。これは、ユーザー King に MyPage、MyWeather または MyContent を表示する権限がないためです。

4. 「**Logout**」をクリックします。

保護されたページへのダイレクト・アクセスの試行

前に、保護されたページに対してログイン認証を強制するセキュリティ制約を定義しました。このため、認証されていないユーザーが保護されたページにアクセスしようとすると、ユーザーは認証のためにログイン・ページにリダイレクトされるはずです。ここで、このことをテストします。

1. アプリケーション・ナビゲータで、「**MyPage.jspx**」を右クリックし、「**実行**」を選択します。

   ブラウザが開いてページが表示されると、ターゲットの URL は次のように表示されます。
   `http://123.4.56.789:8988/MySample-ViewController-context-root/faces/MyPage.jspx`

   これは保護されたページなので、ログイン・フォームが表示されます。

2. 有効なログイン資格証明（`Harvey/welcome` など）を入力し、「**Login**」をクリックします。

   ユーザー Harvey には MyPage を表示する権限があるため、ブラウザに MyPage が表示されます。

3. 「**Logout**」をクリックします。
4. 再度 MyPage へのアクセスを試行します。ブラウザで、手順 1 で使用した URL と同じ URL を入力します。

5. 今度は、MyPage を表示する権限のないユーザー King としてログインします。`King/welcome` を入力し、「**Login**」をクリックします。

   今度は、「Unauthorized」というメッセージが表示されます。

無効な資格証明の入力

チュートリアル・アプリケーションのログイン構成の一部として、ログイン・エラー・ページを選択しました。ここで、無効なユーザー資格証明を入力してみて、動作を確認します。空のログイン・ページに戻されるはずです。

1. アプリケーション・ナビゲータで、「**Welcome.jspx**」を右クリックし、「**実行**」を選択します。
2. ブラウザ・ウィンドウにようこそページが表示されたら、「**Login**」をクリックします。
3. 無効なユーザー資格証明を入力するか、両方のフィールドを空白にしたままにして、「**Login**」をクリックしたときにどのようになるか確認します。

今度は、ようこそページは表示されません。認証されていないユーザーには、空白のログイン・フォームが再び表示されます。

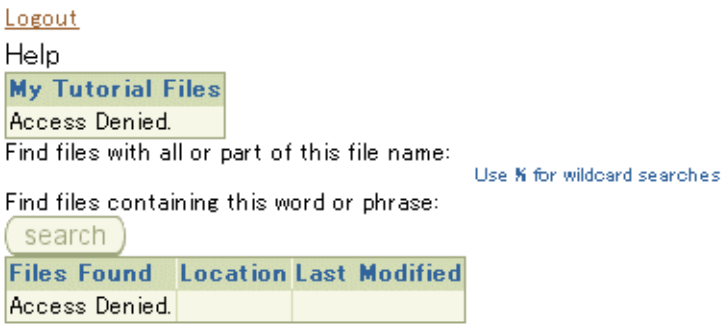
手順 7: データ・コントロールへのアクセスの認可

ページにアクセス制限を適用すると、そのページ上のすべてのデータ・コントロールへのアクセスが自動的に制限されます。ここでは、MyContent.jpsx に配置したデータ・コントロールについて説明します。

1. アプリケーション・ナビゲータで、「**MyContent.jspx**」を右クリックし、「**実行**」を選択します。
2. 有効なログイン資格証明（`Harvey/welcome` など）を入力し、「**Login**」をクリックします。

   ブラウザに MyContent.jspx が表示されますが、今度はファイル・システム・コンテンツにアクセスできません（図 8-51）。

図 8-51 ファイル・システムのデータ・コントロール - アクセス拒否



アクセスを認可するには、データ・コントロールの実行可能ファイルおよびバインディングを介して権限を付与する必要があります。これから、それを行います。

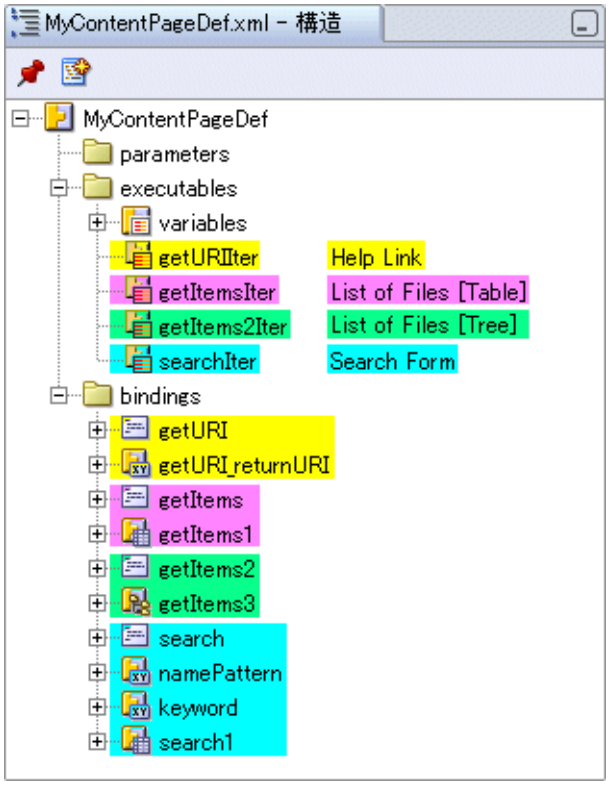
3. アプリケーション・ナビゲータで、「**MyContent.jspx**」を右クリックし、「**ページ定義に移動**」を選択します。

4. 構造ウィンドウで、「**executables**」および「**bindings**」を開きます（図 8-52）。

   データ・コントロールのコンテンツをアクセス可能にするには、コントロールの実行可能ファイル（methodIterator）およびバインディング（methodAction と attributeValues）を介して権限を付与する必要があります。このため、MyContent.jpsx 上にヘルプ・リンクを表示する場合は、`getURIIter`（methodIterator）、`getURI`（methodAction）および `getURI_returnURI`（attributeValues）の認可設定を編集する必要があります。

   同様に、ファイルの表を表示する場合は、`getItemsIter`（methodIterator）、`getItems`（methodAction）および `getItems1`（attributeValues）の認可設定を編集する必要があります。また、ツリーおよび検索フォームのデータ・コントロールに対しても同じ手順を行う必要があります。

図 8-52 MyContentPageDef.xml 内のデータ・コントロールの実行可能ファイルとバインディング



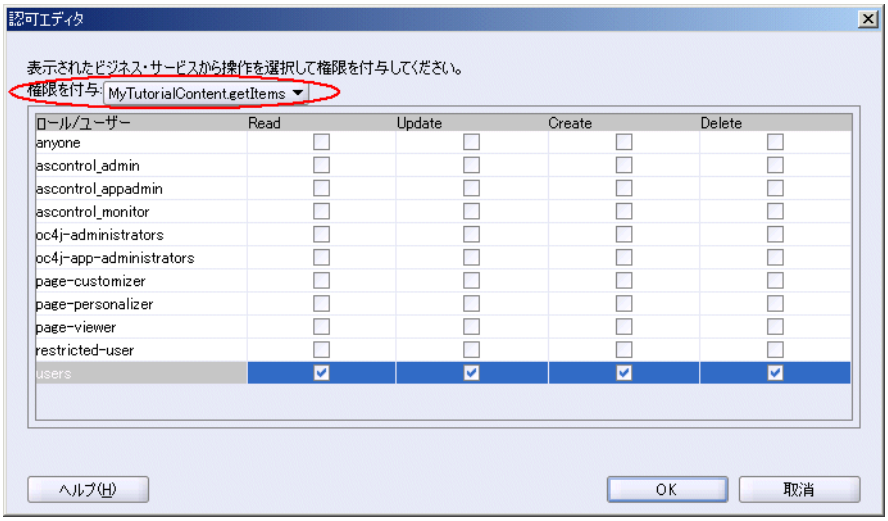
それでは、表データ・コントロールに対するすべての手順を追って、この方法を確認します。

5. まずは、表データ・コントロールの実行可能ファイル getItemsIter の認可を編集します。

   a. 「executables」の下の**「getItemsIter」**を右クリックし、**「認可の編集」**を選択します。

   b. 次に、ログインしたすべてのユーザーに対して完全な権限を付与します。これを行うには、users ロールのすべてのボックスをチェックします（図 8-53 を参照）。

図 8-53 データ・コントロール実行可能ファイルの認可エディタ



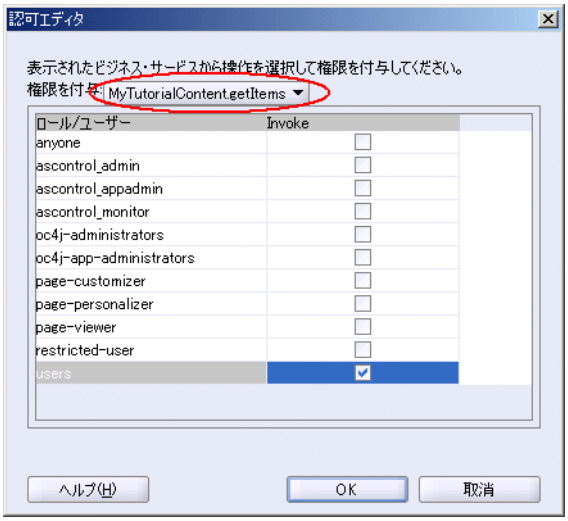
   c. **「OK」**をクリックします。

6. 今度は、表データ・コントロール getItems の methodAction バインディングの認可を編集します。

   a. 「bindings」の下の**「getItems」**を右クリックし、**「認可の編集」**を選択します。

   b. ログインしたすべてのユーザーに権限を付与するには、users ロールのボックスをチェックします（図 8-54 を参照）。

図 8-54 データ・コントロール methodAction バインディングの認可エディタ



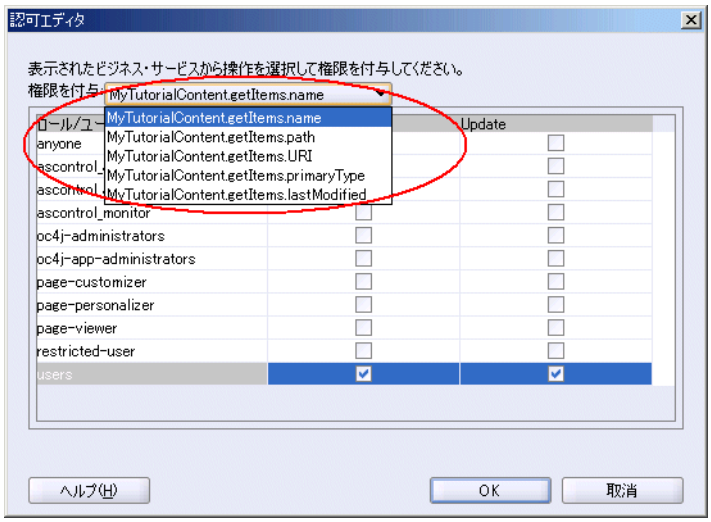
   c. **「OK」**をクリックします。

7. 最後は、表データ・コントロール getItems1 の attributeValues バインディングごとに認可を編集します。

   a. 「**getItems1**」を右クリックし、「**認可の編集**」を選択します。

   b. 最初の属性 **MyTutorialContent.getItems.name** に完全な権限を付与するには、users ロールのすべてのボックスをチェックします（図 8-55 を参照）。

図 8-55 データ・コントロール attributeValues バインディングの認可エディタ



   c. 「権限の付与」ボックスにリストされている属性ごとに、これを繰り返します。たとえば、「権限の付与」ボックスを開き、「MyTutorialContent.getItems.path」を選択して、すべてのボックスをチェックします。これを繰り返します。

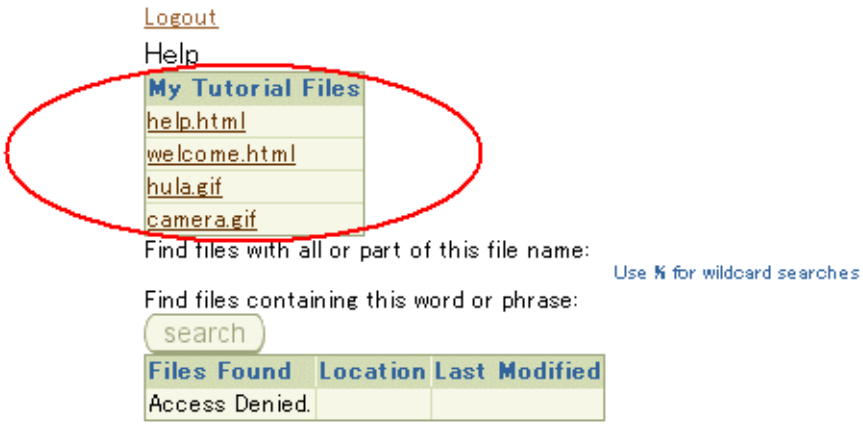
   d. 「**OK**」をクリックします。

8. 再度ページを実行する前に、埋込み OC4J サーバーを停止します。メイン・メニューから「**実行**」、「**終了**」、「**埋込み OC4J サーバー**」の順に選択します。

9. 再度 MyContent.jspx を実行し、`Harvey/welcome` などの有効な資格証明を使用してログインします。

   今度は、表データ・コントロールにファイルがリストされます（図 8-56 を参照）。

図 8-56 表データ・コントロールへのアクセス認可



必要に応じて、手順 5 ～ 7 を繰り返すことにより、MyContent.jspx 上の他のデータ・コントロールへのアクセスを有効化することもできます。

おめでとうございます。これでレッスンは完了です。最初の WebCenter アプリケーションが保護されました。最後のレッスンでは、デプロイメントと、Enterprise Manager（管理者が WebCenter アプリケーションをデプロイ、構成および管理できるブラウザベースのツール）の使用方法について学びます。

9

WebCenter アプリケーションのデプロイ

このレッスンでは、チュートリアル・アプリケーションを WebCenter Preconfigured OC4J にデプロイします。アプリケーションをデプロイするには、必要なすべてのファイルを標準的な J2EE 形式およびディレクトリ構造（WAR ファイルまたは EAR ファイル）でパッケージ化する必要があります。

WebCenter アプリケーションのパッケージ化およびデプロイメントの指示はすべて、デプロイメント・プロファイルを介して構成されています。デプロイメント・プロファイルとは、ページ、ポートレット、カスタマイズ、アプリケーションを構成するメタデータ、作成されるアーカイブ・ファイルのタイプと名前、依存性情報、プラットフォーム固有の手順などが指定された構成ファイルです。

WebCenter アプリケーション専用に、特殊なデプロイメント・プロファイル（WebCenter アプリケーション・デプロイメント・プロファイル）があります。このプロファイルによって、WebCenter アプリケーションに固有のコンテンツ（ポートレット、コンテンツ・リポジトリのデータ・コントロール、カスタマイズ可能コンポーネントなど）が処理されます。

スタンドアロン OC4J と Oracle JDeveloper が同じコンピュータ上にあり、共通のネットワーク・ドライブにアクセスできる場合は、WebCenter アプリケーションを Oracle JDeveloper から直接スタンドアロン OC4J インスタンスにデプロイできます。このレッスンでは、1 つの手順からなるデプロイメントについて詳しく学びます。また、Application Server Control コンソールを介して WebCenter アプリケーションを管理する方法についても学びます。

概要

次の手順を実行することによって、チュートリアル・アプリケーションをデプロイします。

- 手順 1: WebCenter アプリケーション・デプロイメント・プロファイルの作成
- 手順 2: Preconfigured OC4J への直接のデプロイ
- 手順 3: セキュリティ・ポリシーの移行
- 手順 4: デプロイ済アプリケーションの実行
- 手順 5: Application Server Control コンソールを使用した WebCenter アプリケーションの管理

前提条件

WebCenter アプリケーションをデプロイする前に、次の手順を実行します。

1. 次のことを確認します。
   - WebCenter Preconfigured OC4J と Oracle JDeveloper の間に接続が存在している。
   - WebCenter Preconfigured OC4J が稼働中である。

   詳細は、「手順 2: 接続の設定」（第 3 章）を参照してください。
2. チュートリアルのユーザーおよびロールを WebCenter Preconfigured OC4J（デプロイメント・ターゲット）に移行します。JAZN 移行ツールをレルム・モードで実行して、ターゲットの OC4J 上でユーザーおよびロールをマージします。
   a. JAZN 移行ツールを実行する前に、`JDEVHOME¥j2ee¥home¥jazn.jar` および `JDEVHOME¥BC4J¥lib¥adfshare.jar` への参照が含まれるようにクラスパスを更新します。

      たとえば、コマンド・プロンプトで次のように入力します。

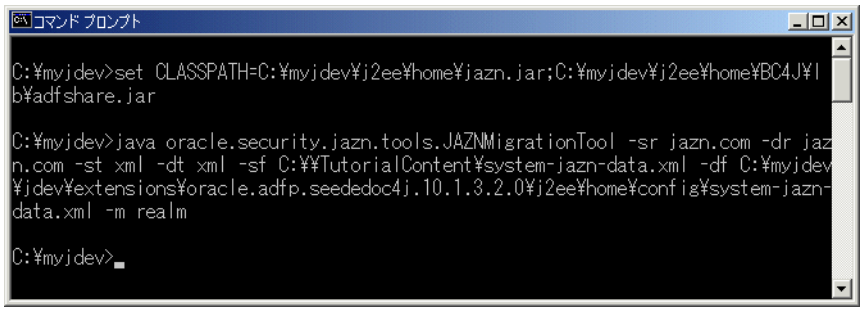
      ```
      set CLASSPATH=JDEVHOME¥j2ee¥home¥jazn.jar;JDEVHOME¥BC4J¥lib¥
      adfshare.jar
      ```

      ここで、*JDEVHOME* は、JDeveloper インストール（たとえば、`C:¥myjdev` など）を指します。
   b. 次に、`JDEVHOME` に移動し、コマンド・プロンプトを開きます。
   c. 次のように入力して、JAZN 移行ツールを実行します。

      ```
      java oracle.security.jazn.tools.JAZNMigrationTool -sr jazn.com
      -dr jazn.com -st xml -dt xml -sf
      WEBCENTERSAMPLE¥TutorialContent¥system-jazn-data.xml -df
      JDEVHOME¥jdev¥extensions¥oracle.adfp.seededoc4j.10.1.3.2.0¥j2ee¥
      home¥config¥system-jazn-data.xml -m realm
      ```

ここで、*WEBCENTERSAMPLE* は、サンプル・チュートリアル・ファイルをインストールした場所（たとえば、`C:` など）を指し、*JDEVHOME* は JDeveloper インストール（たとえば、`C:¥myjdev` など）を指します。図 9-1 を参照してください。

図 9-1 JAZN 移行ツールの実行



これで、Preconfigured OC4J の `system-jazn-data.xml` ファイルには、アプリケーションで必要なユーザーおよびロールが含まれます。

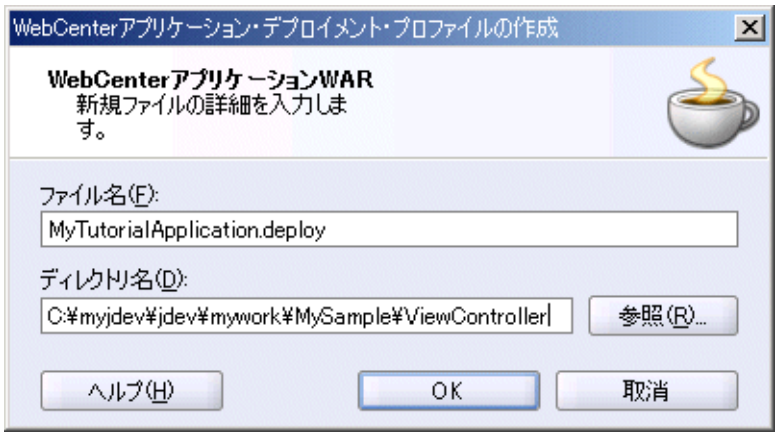
今度は、アプリケーションをデプロイします。

手順 1: WebCenter アプリケーション・デプロイメント・プロファイルの作成

この手順では、`MyTutorialApplication.deploy` という名前のチュートリアル・アプリケーション用のデプロイメント・プロファイルを作成し、いくつかのデプロイメント・オプションを設定します。

1. アプリケーション・ナビゲータで、**「ViewController」**を右クリックし、**「新規」**を選択します。
2. 「General」カテゴリを開き、**「Deployment Profiles」**を選択します。
3. **「WebCenter アプリケーション WAR」**を選択し、**「OK」**をクリックします。
4. デプロイメント・プロファイルの名前を入力します（図 9-2）。「ファイル名」に `MyTutorialApplication` を入力します。

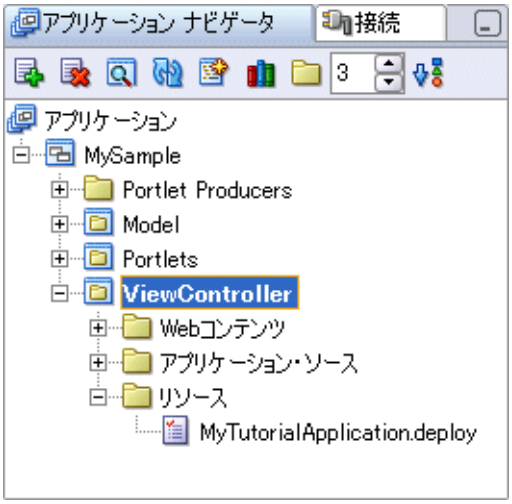
図 9-2 WebCenter アプリケーション・デプロイメント・プロファイルの作成



5. 「**OK**」をクリックします。

   「**ViewController**」、「**リソース**」の下に、新しいデプロイメント・プロファイルが表示されます（図 9-3）。

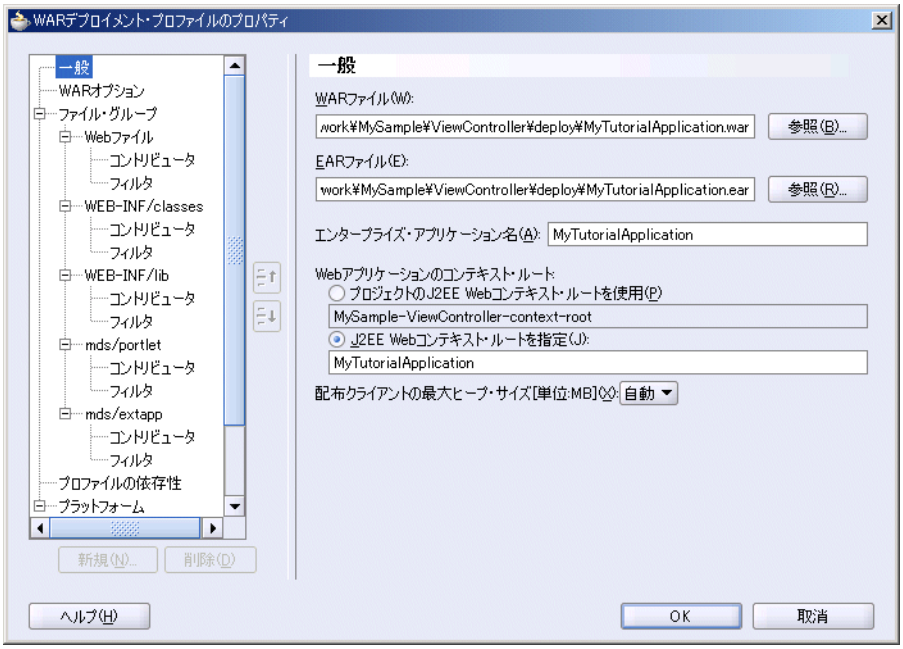
図 9-3 新しいデプロイメント・プロファイル - MyTutorialApplication.deploy



6. デプロイメント・プロファイルを構成するには、アプリケーション・ナビゲータで「**MyTutorialApplication.deploy**」を右クリックし、「**プロパティ**」を選択します。

7. WAR デプロイメント・プロファイルのプロパティ・ウィンドウ（図 9-4）で、アプリケーション URL の適切なコンテキスト・ルートを入力します。「**J2EE Web コンテキスト・ルートを指定**」を選択し、`MyTutorialApplication` を入力します。

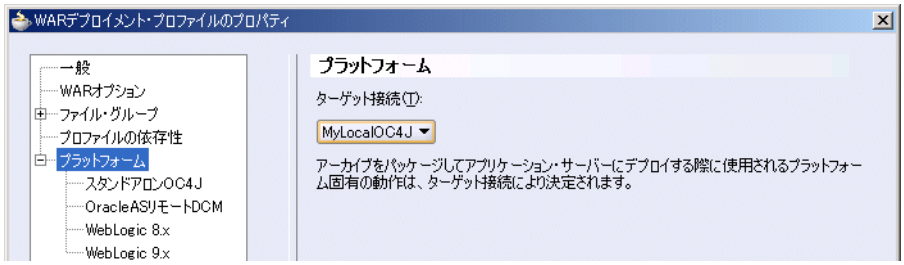
図 9-4 WAR デプロイメント・プロファイルのプロパティ



8. 左側のリストから**「プラットフォーム」**を選択し、「ターゲット接続」ドロップダウン・リストから **MyLocalOC4J** という名前の接続を選択します（図 9-5）。

   この接続は、第 3 章「最初のポートレットの構築およびテスト」で定義したものです。

図 9-5 WAR デプロイメント・プロファイルのプロパティ - ターゲット接続



   このページでは、他のデプロイメント・オプションを設定する必要はありません。アプリケーションに特別なライブラリが必要な場合は、ここでそのライブラリを選択します。

9. **「OK」**をクリックします。

10. JDeveloper のツールバーで、**「すべて保存」**をクリックします。

開発中、1 つの手順で JDeveloper からスタンドアロン OC4J に直接 WebCenter アプリケーションをデプロイできます。次の手順で、この方法を示します。

Application Server Control コンソールを使用して、WebCenter アプリケーションを本番環境にデプロイできます。このプロセスでは、WebCenter アプリケーションを汎用の EAR または WAR ファイル内にパッケージ化します。その後、このファイルに対してデプロイ前ツールを実行して、WebCenter アプリケーションの外部依存性（たとえば、ポートレット・プロデューサのエンド・ポイントと MDS の場所など）を再マップします。デプロイ前ツールによって、ターゲットとなる EAR ファイルが、リモート・システムにデプロイできる状態で生成されます。さらに、セキュリティ、コンテンツ統合および外部アプリケーションに関連するいくつかのタスクを、デプロイメント・プロセスの一部として実行する必要があることがあります。

WebCenter アプリケーション・デプロイメントの詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

手順 2: Preconfigured OC4J への直接のデプロイ

この手順では、前に定義した接続詳細を使用して、チュートリアル・アプリケーションを直接 Preconfigured OC4J にデプロイします。これから、それを行います。

1. 「**MyTutorialApplication.deploy**」を右クリックし、「**MyLocalOC4J に配布**」を選択します。

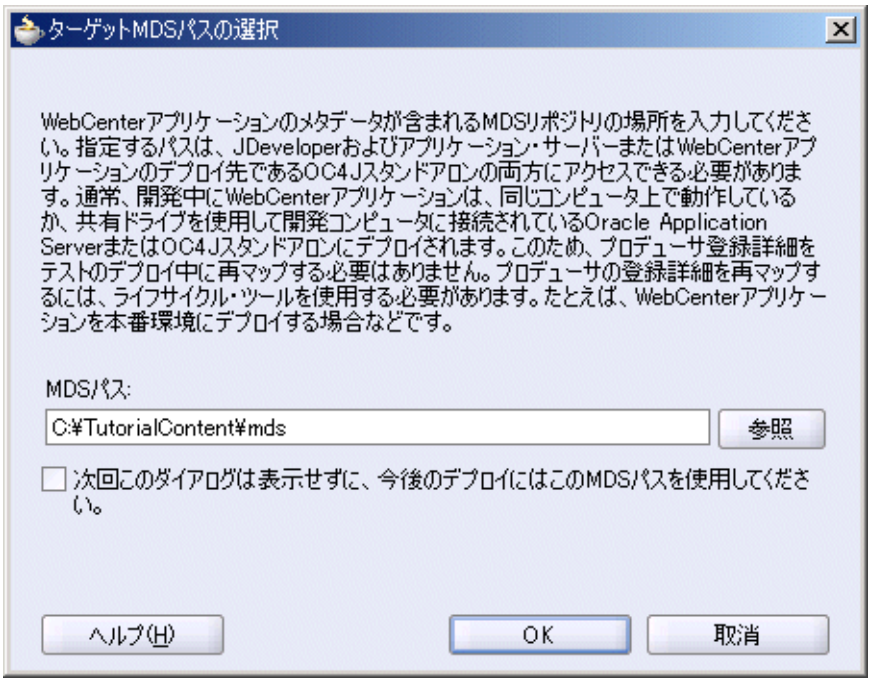
   ダイアログで、ファイル・システム内に MDS リポジトリの適切な場所を定義するように要求されます（図 9-6）。

2. 「MDS パス」に `C:¥TutorialContent¥mds` を入力します。

   必要に応じて、ローカル・ファイル・システム上の別の場所を選択することもできます。指定したディレクトリが存在しない場合は、JDeveloper によって自動的にそのディレクトリが作成されます。

注意： デプロイメント用の MDS リポジトリは、アプリケーションの作業ディレクトリ（この場合は `JDEVHOME¥jdev¥mywork¥MySample`）内に作成**しないで**ください。開発用とデプロイメント用に別々の MDS ディレクトリを保持すれば、混同を避け、アプリケーションのソース MDS リポジトリ内のコンテンツが上書きされることも避けられます。

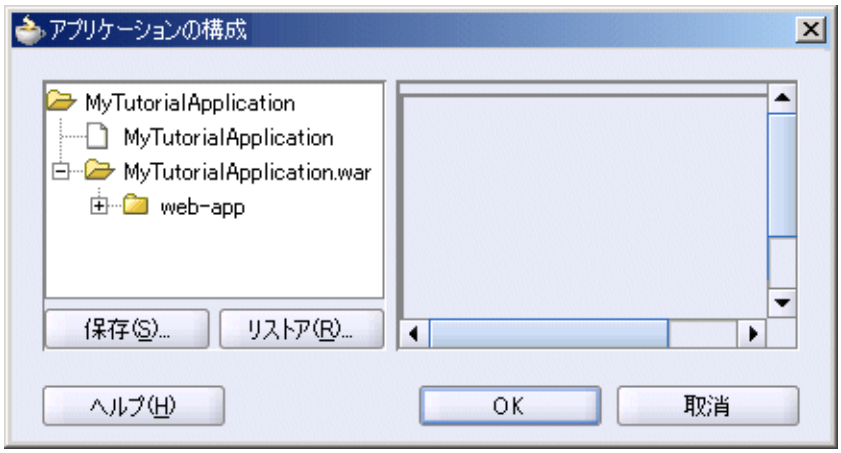
図 9-6 ターゲットの MDS パス



3. 「**OK**」をクリックします。

4. アプリケーションの構成ウィンドウ（図 9-7）が表示されるのを待ってから、「**OK**」をクリックします。

図 9-7 アプリケーションの構成ウィンドウ



WAR ファイルおよび EAR ファイルが生成されて、Preconfigured OC4J にデプロイされます。EAR ファイルには、WAR ファイルおよび多数の構成ファイル（.xml）が含まれます。WAR ファイルには、アプリケーションによって使用されるすべてのファイルとライブラリが含まれます。

デプロイメントの終了時に、「デプロイ - ログ」ウィンドウに「デプロイが終了」と表示されます。このメッセージが表示されたら、デプロイ済アプリケーションにアクセスできるようになりますが、ログインはできません。まださらに、（`app-jazn-data.xml` に格納されている）アプリケーションのセキュリティ・ポリシーを Preconfigured OC4J の `system-jazn-data.xml` ファイルに移行する必要があります。これから、それを行います。

手順 3: セキュリティ・ポリシーの移行

この手順では、JAZN 移行ツールを使用して、（`app-jazn-data.xml` に格納されている）アプリケーションのセキュリティ・ポリシーを Preconfigured OC4J の `system-jazn-data.xml` ファイルに移行します。

注意： JAZN 移行ツールの詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

このチュートリアルでは、これらの XML ファイルの場所は次のとおりです。

ソースの app-jazn-data.xml

```
JDEVHOME¥jdev¥extensions¥oracle.adfp.seededoc4j.10.1.3.2.0¥j2ee¥home¥
applications¥MyTutorialApplication¥adf¥META-INF¥app-jazn-data.xml
```

移行先の system-jazn-data.xml

```
JDEVHOME¥jdev¥extensions¥oracle.adfp.seededoc4j.10.1.3.2.0¥j2ee¥home¥
config¥system-jazn-data.xml
```

次の手順を実行します。

1. JAZN 移行ツールを実行する前に、次の .JAR ファイルへの参照が含まれるようにクラスパスを更新します。

   - JDEVHOME¥j2ee¥home¥jazn.jar
   - JDEVHOME¥BC4J¥lib¥adfshare.jar

   たとえば、コマンド・プロンプトで次のように入力します。

   ```
   set CLASSPATH=JDEVHOME¥j2ee¥home¥jazn.jar;JDEVHOME¥BC4J¥lib¥
   adfshare.jar
   ```

   ここで、*JDEVHOME* は、JDeveloper インストール（たとえば、`C:¥myjdev` など）を指します。

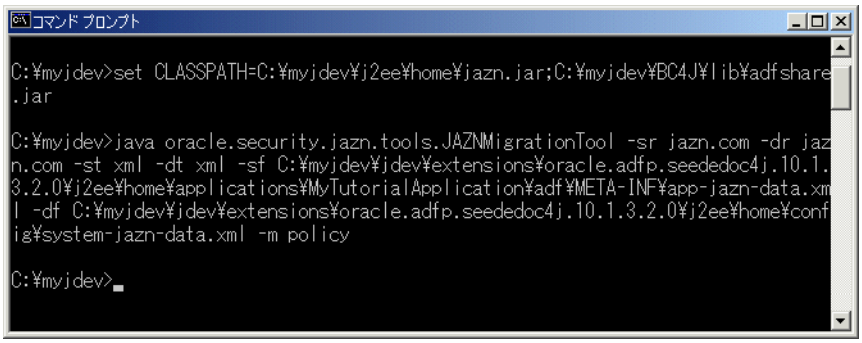
2. 次に、`JDEVHOME` に移動し、コマンド・プロンプトを開きます。

3. 次のように入力して、JAZN 移行ツールを実行します。

   ```
   java oracle.security.jazn.tools.JAZNMigrationTool -sr jazn.com -dr
   jazn.com -st xml -dt xml -sf
   JDEVHOME¥jdev¥extensions¥oracle.adfp.seededoc4j.10.1.3.2.0¥j2ee¥
   home¥applications¥MyTutorialApplication¥adf¥META-INF¥
   app-jazn-data.xml -df
   JDEVHOME¥jdev¥extensions¥oracle.adfp.seededoc4j.10.1.3.2.0¥j2ee¥
   home¥config¥system-jazn-data.xml -m policy
   ```

   ここで、*JDEVHOME* は、JDeveloper インストール（たとえば、`C:¥myjdev` など）を指します。例は、図 9-8 を参照してください。

図 9-8 JAZN 移行ツールの実行



これで、Preconfigured OC4J の `system-jazn-data.xml` ファイルにはアプリケーションのセキュリティ情報が含まれます。この情報を使用可能にするには、WebCenter Preconfigured OC4J を再起動する必要があります。

4. JDeveloper から WebCenter Preconfigured OC4J を再起動するには、次のようにします。

   a. JDeveloper ツールバーの一番右にある**「WebCenter Preconfigured OC4J の停止」**をクリックします。

   b. **「WebCenter Preconfigured OC4J の起動」**アイコンをクリックします。

   OC4J インスタンスが初期化されたことを示すメッセージを待ちます。

今度は、デプロイ済アプリケーションのページの 1 つを表示してみます。

手順 4: デプロイ済アプリケーションの実行

1. ブラウザ・ウィンドウを開き、チュートリアルのようこそページにナビゲートします。デプロイメント・プロファイル（MyTutorialApplication）内に定義されているコンテキスト・ルートを使用する URL を入力し、続いて /faces/、ページ名の順に入力する必要があります。必要な URL 書式は次のとおりです。

   ```
   http://<host>:<port>/<context-root>/faces/<page-name>
   ```

   次に例を示します。

   ```
   http://localhost:6688/MyTutorialApplication/faces/Welcome.jspx
   ```

   このチュートリアルでは、URL 内に `localhost` を使用すると、Preconfigured OC4J がインストールされているローカル・コンピュータを参照できることを前提としています。（そうでない場合は、`localhost` をご使用のコンピュータの IP アドレスに置き換えてください。）

2. ユーザー `Harvey` としてログインし、パスワード `welcome` を入力します。

3. 各ページ・ナビゲーション・ボタンをクリックして、アプリケーションが予測どおりに動作することを確認します。

おめでとうございます。これで、最初の WebCenter アプリケーションのデプロイが完了しました。

注意： WebCenter Preconfigured OC4J は、このチュートリアルのようなテスト・デプロイメントにのみ適しています。通常は、ポートレット・プロデューサに使用しているのと同じ OC4J インスタンスに WebCenter アプリケーションをデプロイします。OC4J を Oracle Application Server またはスタンドアロンの OC4J インスタンスにデプロイする方法は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

手順 5: Application Server Control コンソールを使用した WebCenter アプリケーションの管理

この手順では、Oracle Enterprise Manager 10*g* Application Server Control コンソールを使用して WebCenter アプリケーションをデプロイ、再デプロイおよびアンデプロイする方法を学びます。また、コンソールからポートレット・パフォーマンスを監視する方法についても学びます。

1. Preconfigured OC4J 用の Application Server Control コンソールにアクセスするには、次の URL にナビゲートします。

   ```
   http://<host>:<port>/em
   ```

   たとえば、`http://localhost:6688/em` にナビゲートします。

   このチュートリアルでは、URL 内に `localhost` を使用すると、Preconfigured OC4J がインストールされているローカル・コンピュータを参照できることを前提としています。（そうでない場合は、`localhost` をご使用のコンピュータの IP アドレスに置き換えてください。）

   詳細は、JDeveloper のメイン・メニューから、**「ヘルプ」**、**「WebCenter Preconfigured OC4J README」**の順に選択します。

2. ユーザー `oc4jadmin` としてログインします。

3. デフォルト・パスワード `welcome` を入力します。

   Preconfigured OC4J のホームページが表示されます。

4. **「アプリケーション」**タブを選択し、アプリケーション名を選択します。アプリケーション名は、.EAR ファイル名から導出されます。このチュートリアルでは、.EAR ファイルは、デプロイメント・プロファイル（**MyTutorialApplication**）に基づいて名前が付けられています。

5. 「**MyTutorialApplication**」をクリックすると、図 9-9 に示すようなアプリケーションのホームページが表示されます。

図 9-9 Enterprise Manager での WebCenter アプリケーションのホームページ

このページから、ボタンを使用してアプリケーションを再デプロイおよびアンデプロイできます。また、アプリケーションの停止および再起動もこのページから実行できます。

「**関連リンク**」の下に、「**プロデューサおよびポートレット**」リンクが表示されています。ここから使用可能な情報は、WebCenter アプリケーションに固有のものであり、プロデューサおよびポートレットのメトリックが使用可能な場合にのみ表示されます。統計情報は、ポートレットを含むページが初めてアクセスされたときに使用可能になります。「手順 4: デプロイ済アプリケーションの実行」のときに、ポートレットを含むページが表示されなかった場合は、ここで表示し（MyPage または MyWeather を開く）、Application Server Control コンソールをリフレッシュします。

6. **「プロデューサおよびポートレット」**をクリックして、チュートリアル・アプリケーションに使用可能なメトリックを参照します。

   (図 9-10 に示すような)「ポートレット・プロデューサ」ページが表示されます。このページから、WebCenter アプリケーションで使用されるプロデューサおよびポートレットのステータスとパフォーマンスを監視できます。

図 9-10 Enterprise Manager でのポートレット・プロデューサ画面

Application Server Control コンソールから WebCenter アプリケーションをデプロイ、構成および監視する方法の詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

まとめ

このチュートリアルでは、簡単なセキュア WebCenter アプリケーションを構築しました。また、このアプリケーションをテストしてデプロイし、Application Server Control コンソールを使用してポートレット・プロデューサを監視しました。

Oracle JDeveloper および Oracle WebCenter Framework の機能についての知識が深まったので、これで独自のアプリケーションの構築を開始できます。詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

第 III 部

付録

第 III 部の内容は次のとおりです。

A

チュートリアルの ID ストアの設定方法

この付録では、第 8 章「セキュリティの設定」の要件である ID ストアを設定する方法について説明します。

Oracle ADF Security では、特定のリソース・プロバイダに対してユーザーが認証されます。このチュートリアルでは、埋込み OC4J に付属の軽量 XML リソース・プロバイダ `system-jazn-data.xml` を活用します。このリソース・プロバイダは、このチュートリアルのような小規模なアプリケーションに最適であり、`JDEVHOME¥jdev¥system¥oracle.j2ee.10.1.3.xx.xx¥embedded-oc4j¥config` にあります。

> **注意：** JDeveloper を最初に開いたときに、システム・ディレクトリが作成されます。

便宜上、このチュートリアルを完了するのに必要なすべてのデータが含まれたサンプルの `system-jazn-data.xml` ファイルが用意されています（第 2 章の「サンプル・チュートリアル・ファイルのダウンロード」および「サンプルの system-jazn-data.xml ファイルのコピー」を参照）。次の表に、サンプル・ファイルで提供されているユーザーおよびロールについて概説します。

ロール名	ユーザー	説明
page-viewer	Singh	このユーザーは、保護されたページを表示できます。
page-personalizer	Cho	このユーザーは、保護されたページ上のポートレットをパーソナライズできます。
page-customizer	Harvey	このユーザーは、保護されたページをカスタマイズできます。
restricted-user	King	このユーザーは、保護されたページを表示できません。
users	Singh、Cho、King、Harvey、JtaAdmin、oc4jadmin	users ロールは、有効な各ユーザーのリストを保守します。

これらのチュートリアル・ユーザーおよびロールを最初から自分で入力する場合のみ、この付録の手順に従ってください。プロセスの詳細を知りたい場合や、すでに JDeveloper を使用してセキュアなアプリケーションを構築中の場合は、追加したユーザー、ロールおよびポリシーを上書きする必要はありません。

ID ストアを設定するには、次の各項の手順を実行します。

- ユーザーの作成
- ロールの作成およびユーザー・メンバーの割当て
- チュートリアル・ユーザーおよびロールを JDeveloper の認可エディタで使用可能にする

ユーザーの作成

この手順では、Singh、Cho、Harvey および King という 4 つのユーザーを埋込み OC4J の `system-jazn-data.xml` ファイルに追加します。

1. 「**ツール**」メニューから、「**埋込み OC4J サーバーの設定**」を選択します。

   「現在実行中の埋込みサーバー」という情報メッセージが表示されたら、「**いいえ**」をクリックし、埋込み OC4J サーバーを停止（メイン・メニューから「**実行**」、「**終了 - 埋込み OC4J サーバー**」の順に選択）します。

2. 「**グローバル**」ブランチの下の「**認証 (JAZN)**」、「**レルム**」、「**jazn.com**」を順に開きます。

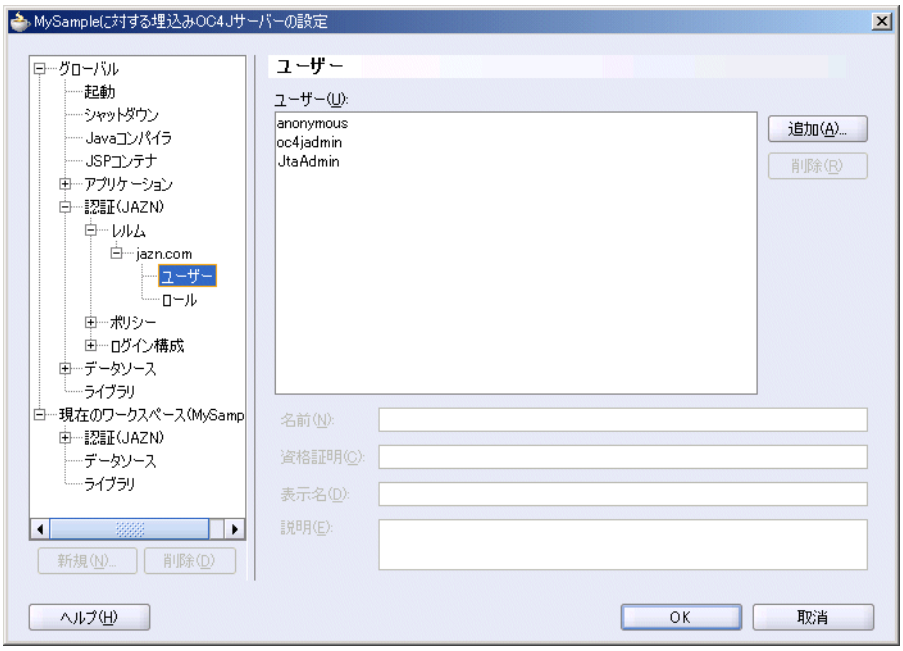
   jazn.com は、チュートリアル・アプリケーションのデフォルトのセキュリティ・レルムです。

   「**現在のワークスペース**」の下の「**認証 (JAZN)**」というブランチを選択しないでください。このブランチではアプリケーション・レベルでユーザー・データを定義できますが、チュートリアル・アプリケーションでは使用されません。WebCenter アプリケーションでは、「グローバル」レルムの下に定義されているデータのみが使用されます。

3. 「**ユーザー**」を選択します。

   デフォルトのグローバル・セキュリティ・レルムに対する 3 つの事前定義ユーザーが表示されています（図 A-1）。

図 A-1 グローバル・セキュリティ・レルム jazn.com のデフォルト・ユーザー



3 つのデフォルト・ユーザーは、次のとおりです。

- **anonymous:** デフォルトのゲストまたは匿名ユーザー
- **oc4jadmin:** OC4J 管理者
- **JtaAdmin:** 伝播済 OC4J トランザクションをリカバリするための別のユーザー

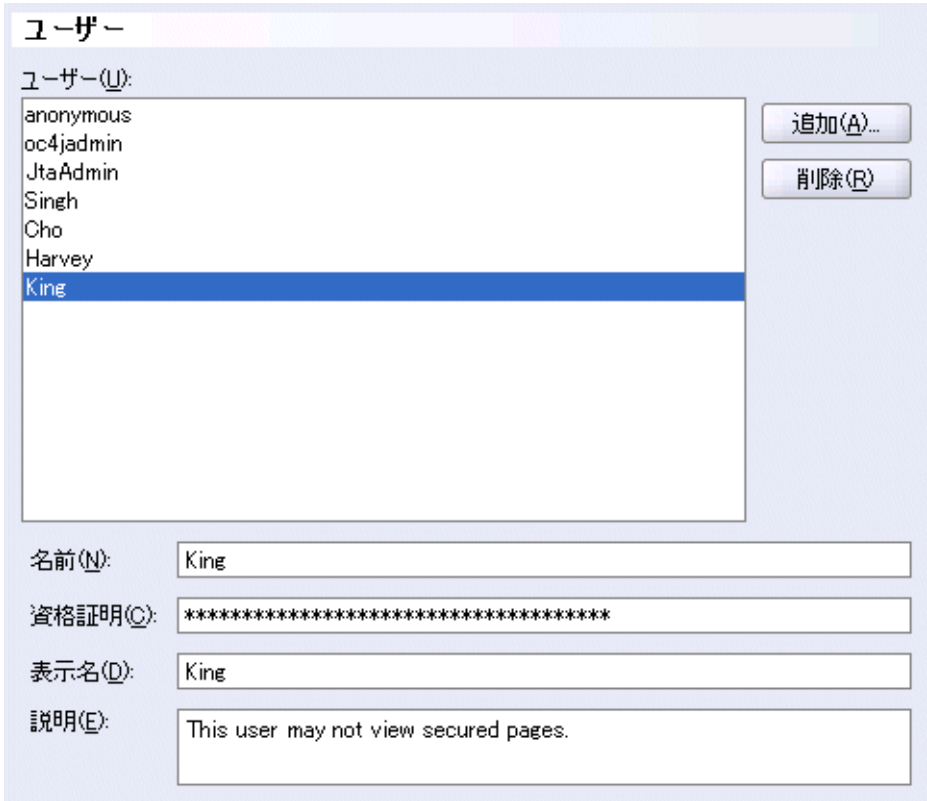
これらのユーザーはいずれも削除しないでください。削除すると、管理機能の一部が働かなくなります。

4. Singh という名前の新しいユーザーを作成します。
   a. **「追加」**をクリックします。
   b. 「名前」に `Singh` を入力します。
   c. 「資格証明」フィールドに、パスワード `welcome` を入力します。
   d. **「OK」**をクリックします。「ユーザー」リストに Singh が表示されます。
   e. 「説明」に、`This User may view pages` を入力します。
5. 次に、手順 4 を繰り返します。Cho、Harvey および King という名前の 3 つのユーザーを追加で作成します。次の表に示す資格証明および説明を使用してください。

名前	資格証明	表示名	説明
Singh	welcome	Singh	This user may view secured pages.
Cho	welcome	Cho	This user may personalize portlets on a secured page.
Harvey	welcome	Harvey	This user may customize secured pages.
King	welcome	Harvey	This user may not view secured pages.

「ユーザー」リストに、新しい 4 つのユーザーが表示されます（図 A-2 を参照）。

図 A-2 新しいユーザー



6. **「OK」**をクリックして、埋込み OC4J の `system-jazn-data.xml` にユーザー定義を保存します。

ロールの作成およびユーザー・メンバーの割当て

この手順では、page-viewer、page-personalizer、page-customizer および restricted-user という 4 つのロールを埋込み OC4J の `system-jazn-data.xml` ファイルに追加します。

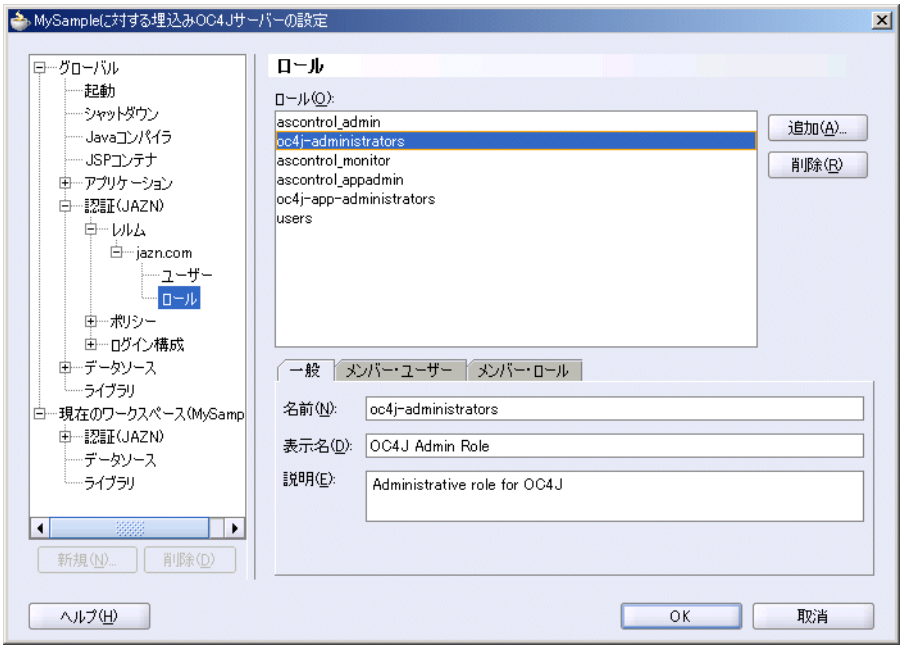
1. 「**ツール**」メニューから、「**埋込み OC4J サーバーの設定**」を選択します。
2. 「**認証 (JAZN)**」、「**レルム**」および「**jazn.com**」を開きます。
3. 「**ロール**」を選択します。

   デフォルトのグローバル・セキュリティ・レルム jazn.com に対する事前定義ロールがいくつか表示されます（図 A-3 を参照）。

   - **oc4j-administrators:** OC4J 管理者ロール
   - **oc4j-app-administrators:** OC4J アプリケーション管理者ロール
   - **users:** システム内のすべてのユーザーをマップするための汎用グループ
   - **ascontrol_admin:** Enterprise Manager Application Server Control 管理者ロール
   - **ascontrol_appadmin:** Enterprise Manager アプリケーション管理者ロール
   - **ascontrol_monitor:** Enterprise Manager 監視ロール

   これらのロールはいずれも削除しないでください。削除すると、管理機能の一部が働かなくなります。詳細は、『Oracle Application Server 管理者ガイド』を参照してください。

図 A-3 グローバル・セキュリティ・レルム jazn.com のデフォルト・ロール



4. page-viewer という名前の新しいロールを作成し、ユーザー Singh をこのロールに割り当てます。
   a. 「**追加**」をクリックします。
   b. 「名前」に `page-viewer` を入力し、「**OK**」をクリックします。
   c. 「**メンバー・ユーザー**」タブをクリックし、**Singh** を右側のリストに移動します。

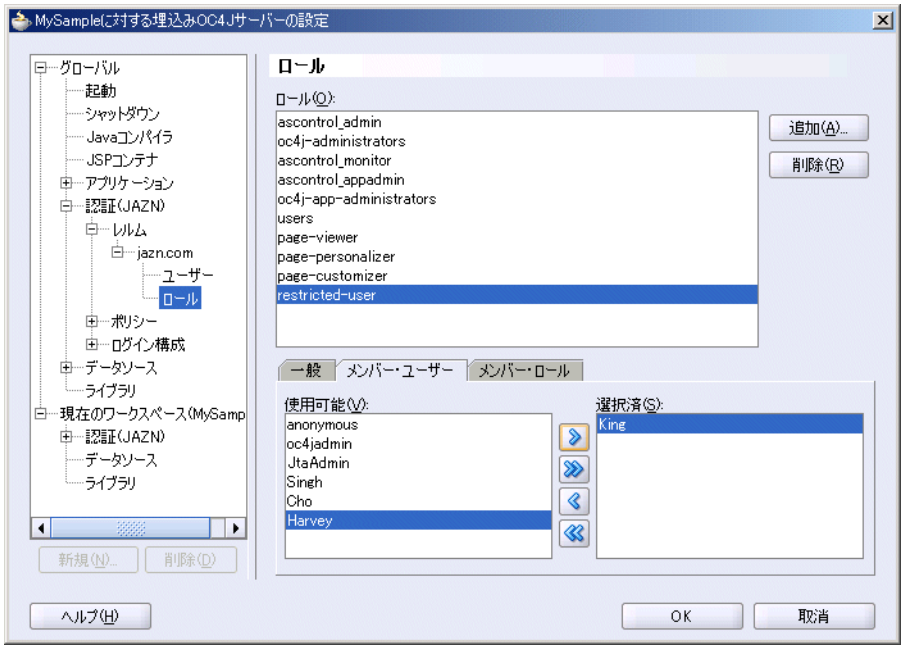
5. 次に、手順 4 を繰り返します。さらに 3 つのロールを追加し、この表に示すとおりに、メンバー・ユーザーを各ロールに割り当てます。

ロール	メンバー・ユーザー
page-viewer	Singh
page-personalizer	Cho
page-customizer	Harvey
restricted-user	King

   a. page-personalizer、page-customizer および restricted-user という名前のロールを追加します。

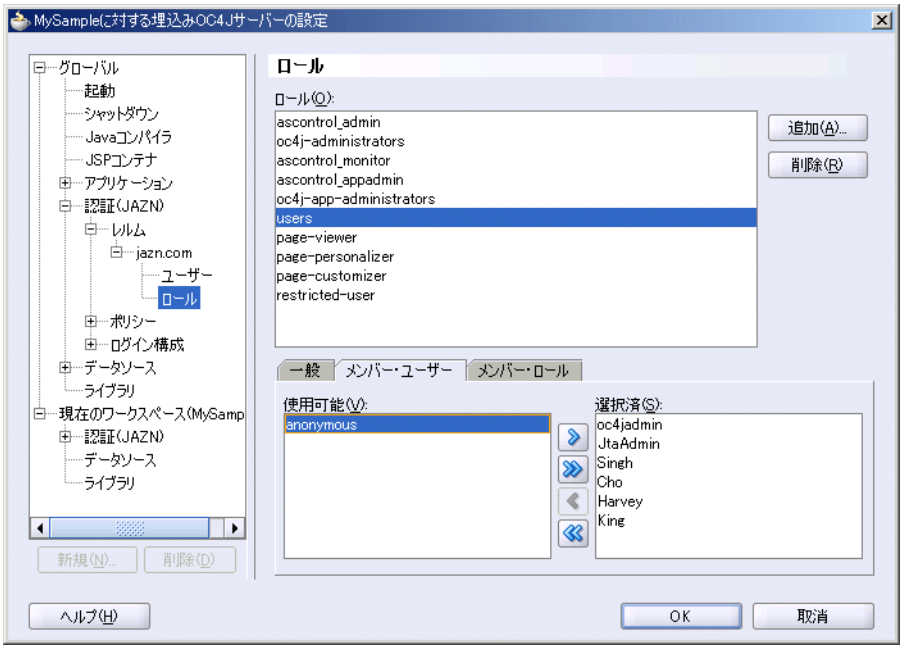
   b. メンバー Cho を page-personalizer ロールに、メンバー Harvey を page-customizer ロールに、メンバー King を restricted-user ロールに割り当てます（図 A-4 を参照）。

図 A-4 新しいロールに割り当てられたメンバー・ユーザー



6. （「anonymous」を除く）すべてのユーザーを users ロール内に配置します。
   a. 「**users**」ロールを選択します。
   b. 「**メンバー・ユーザー**」タブをクリックし、ユーザー（Singh、Cho、Harvey、King、JtaAdmin および oc4jadmin）を右側のリストに移動します（図 A-5 を参照）。

図 A-5 users ロールへのメンバーの割当て



`users` ロールは、有効な各ユーザーのリストを保守します。第 8 章「セキュリティの設定」では、このロールを `ValidUsers` という名前の J2EE セキュリティ・ロールにマップします（詳細は「手順 2: ADF Security 設定の構成」を参照）。

7. 「**OK**」をクリックして、埋込み OC4J の `system-jazn-data.xml` ファイルにロール定義を保存します。

次の手順では、Oracle JDeveloper の認可エディタを介してこれらのユーザーおよびロールを使用可能にします。第 8 章の「手順 4: ページの保護」で、このエディタを介してページ権限を割り当てます。

チュートリアル・ユーザーおよびロールを JDeveloper の認可エディタで使用可能にする

この手順では、チュートリアルのユーザーおよびロールを JDeveloper のホーム・ディレクトリにコピーして、JDeveloper デザインタイム・ダイアログで使用可能にします。

1. このチュートリアルの目的で変更を行う前に、JDEVHOME¥j2ee¥home¥config にある `system-jazn-data.xml` ファイルをバックアップしてください。
2. `system-jazn-data.xml` ファイルを、埋込み OC4J ディレクトリ `JDEVHOME¥jdev¥system¥oracle.j2ee.10.1.3.xx.xx¥embedded-oc4j¥config` から `JDEVHOME¥j2ee¥home¥config` にコピーします。

注意： この場所にすでに `system-jazn-data.xml` ファイルを移入してある場合は、元のファイルを上書きせずに、ファイルをマージする必要があります。JAZN 移行ツールをレルム・モードで実行して、ユーザーおよびロールをマージします。

1. まず、CLASSPATH を `JDEVHOME¥j2ee¥home¥jazn.jar;JDEVHOME¥BC4J¥lib¥adfshare.jar` に設定します。
2. 次の構文を使用して、JAZN 移行ツールを実行します。

```
java oracle.security.jazn.tools.JAZNMigrationTool -sr
jazn.com -dr jazn.com -st xml -dt xml -sf
JDEVHOME¥jdev¥system¥oracle.j2ee.10.1.3.xx.xx¥embedded-oc
4j¥config¥system-jazn-data.xml -df
JDEVHOME¥j2ee¥home¥config¥system-jazn-data.xml -m realm
```

ここで、*JDEVHOME* は JDeveloper インストール（たとえば、`C:¥myjdev` など）を指し、*10.1.3.xx.xx* はバージョン番号を指します。詳細は、『Oracle WebCenter Framework 開発者ガイド』を参照してください。

索引

N

O

P

R

S

W

X

い

う

ほ

ゆ

よ

ら

り

る

ろ